

# **XPRESS-MP**

## **USER GUIDE**

© **Dash Associates**  
**1999**

All trademarks referenced in this manual that are not the property of Dash Associates are acknowledged.

All companies, products, names and data contained within this user guide are completely fictitious and are used solely to illustrate the use of XPRESS-MP. Any similarity between these names or data and reality is purely coincidental.

### **How to Contact Dash**

If you have any questions or comments on the use of XPRESS-MP, please contact Dash technical support at:

Dash Associates Limited  
Quinton Lodge  
Binswood Avenue  
Leamington Spa  
Warwickshire CV32 5TH  
UK

Telephone: +44 1926 315862  
Fax: +44 1926 315854  
e-mail: [support@dash.co.uk](mailto:support@dash.co.uk)

If you have any sales questions or wish to order XPRESS-MP software, please contact your local sales office or Dash sales at:

Dash Associates Limited  
Blisworth House  
Church Lane  
Blisworth  
Northants NN7 3BX  
UK

Telephone: +44 1604 858993  
Fax: +44 1604 858147  
e-mail: [sales@dash.co.uk](mailto:sales@dash.co.uk)

For the latest news and XPRESS-MP software updates please visit the Dash website at:

<http://www.dash.co.uk/>

---

# Contents

<b>1</b>	<b>An Introduction to XPRESS-MP</b>	<b>1</b>
1.1	What XPRESS-MP Does and Why You Need It .....	1
1.2	What You Need to Know Before Using XPRESS-MP .....	1
1.3	Hardware Platforms Supported by XPRESS-MP .....	2
1.4	Symbols and Conventions .....	2
1.5	Running XPRESS-MP .....	3
1.6	The Structure of this Guide .....	4
<b>2</b>	<b>Getting Started with XPRESS-MP</b>	<b>7</b>
2.1	Entering a Model .....	7
2.2	The Chess Set Problem: Description .....	7
2.2.1	A First Formulation .....	8
2.3	Solving the Chess Set Problem .....	9
2.3.1	Building the model .....	9
2.3.2	Obtaining a Solution using XPRESS-MP .....	12
2.3.3	Interpreting the Output .....	13
2.4	Summary .....	15
<b>3</b>	<b>Two Simple Examples</b>	<b>17</b>
3.1	Example 1: A Simple Production Planning Exercise .....	17
3.1.1	The Model Background .....	18
3.1.2	Developing the Model .....	18
3.1.3	Correcting Syntax Errors in the Model .....	20
3.1.4	Obtaining the Solution .....	22
3.2	Example 2: A Blending Example .....	22
3.2.1	The Model Background .....	23
3.2.2	Developing the Model .....	23
3.2.3	Obtaining the Solution .....	26
3.2.4	Re-running the Model with New Data .....	26
<b>4</b>	<b>The Coco Problem</b>	<b>27</b>
4.1	Overview .....	27
4.2	Problem Description .....	27
4.3	Phase 1: A First Formulation .....	28
4.3.1	Specifying the Variables .....	28
4.3.2	Specifying the Constraints .....	28
4.3.3	Using XPRESS-MP to Solve the Coco Problem .....	30

---



4.3.4 Viewing the Solution.....	31
4.3.5 Where are We?.....	34
4.4 Phase 2: Using Subscripts .....	34
4.5 Phase 3: Multi-Period Modelling.....	39
4.6 Phase 4: Introducing Fixed Costs .....	46
4.7 Phase 5: Permanent Closure? .....	48
4.8 Where Have We Got To?.....	49
<b>5 More Advanced Modelling Features</b> .....	<b>51</b>
5.1 Overview.....	51
5.2 Conditional Generation of Variables and Constraints.....	51
5.3 Importing Data .....	53
5.3.1 The DATA Section .....	53
5.3.2 The DISKDATA Section .....	54
5.3.3 Helpful Tip: Sizing TABLES for Spreadsheet Data.....	56
5.3.4 The ASSIGN Section .....	58
5.4 Sparse Data Tables.....	59
5.5 Displaying Data.....	60
5.5.1 The PRINT Section .....	61
5.5.2 Using DISKDATA to Output Data .....	62
5.6 Index Sets.....	63
5.6.1 Example 1: Using index set with data tables and variables.....	63
5.6.2 Example 2: Index sets in the Coco problem.....	64
5.6.3 Example 3: Mixing index sets and numerical indices.....	68
5.7 Built-In Functions.....	71
5.8 Exporting Results .....	72
5.8.1 Advanced Techniques .....	75
5.9 Case Management.....	75
5.10 Macros .....	76
<b>6 Integer Programming in XPress-MP</b> .....	<b>79</b>
6.1 Introduction to Integer Programming.....	79
6.2 A Project Planning Model .....	81
6.3 A Project Planning Model Using Special Ordered Sets .....	84
<b>Glossary of Terms</b> .....	<b>87</b>
<b>Index</b> .....	<b>93</b>

---

# 1 An Introduction to XPRESS-MP

## 1.1 What XPRESS-MP Does and Why You Need It

XPRESS-MP is a set of powerful mathematical modelling and optimisation tools, available as: **Console XPRESS** – standalone modules `mp-model` and `mp-opt` with text 'console' interfaces; **Visual XPRESS** – an integrated modelling and optimisation system under Windows; or the **XPRESS Libraries** – callable modelling and optimisation subroutine libraries.

In this User Guide remarks that are only applicable to **Visual XPRESS** are signified by , whilst remarks that are applicable only to **Console XPRESS** (`mp-model` and `mp-opt`) are designated thus: . The text to which the symbol applies runs until the normal margins are restored. The use of the **XPRESS Libraries** is beyond the scope of this User Guide, which is about the common XPRESS-MP modelling language and the basic optimiser functionality.

The modelling component provides you with an easy to use yet powerful language for describing your problem. It also enables you to gather the problem data from ASCII files and a range of popular spreadsheets and databases. The optimisation component provides a solver that reads in the problem description produced by the modeller, and then solves it to provide you with the optimal solution. In **Visual XPRESS** the interface between the modelling component and the solution process is hidden from you, and happens automatically.

Broadly speaking, you specify your problem to the model builder and the optimiser actually solves your problem. Unlike conventional programming languages such as C or FORTRAN, XPRESS-MP is non-procedural. You only specify to XPRESS-MP what you want to do and it takes care of how to do it using a powerful set of default actions. Users solving difficult problems can tune XPRESS-MP by changing these defaults. The subroutine libraries can be used in the creation of embedded applications.

## 1.2 What You Need to Know Before Using XPRESS-MP

Before using XPRESS-MP you should be comfortable with the use of symbols such as  $x$  or  $y$  to represent unknown quantities, and the use of this sort of variable in simple linear equations and linear inequalities. For example:

$$x + y \leq 6.$$

---

Experience of a basic course in Mathematical or Linear Programming is recommended but is not essential. Similarly some familiarity with the use of computers would be helpful.

For all but the simplest models you should also be familiar with the idea of summing over a range of variables. For example, if  $x_j$  is used to represent the number of cars produced on production line  $j$  then the total number of cars produced on all  $N$  production lines can be written as:

$$\sum_{j=1}^N x_j$$

This says "sum the output from each production line  $x_j$  over all production lines  $j$  from  $j=1$  to  $j=N$ ". If our target is to produce at least 1000 cars in total then we would write the inequality:

$$\sum_{j=1}^N x_j \geq 1000$$

The XPRESS-MP modelling language closely mimics the mathematical notation an analyst uses to describe a problem. So provided you are happy using the above mathematical notation the step to using XPRESS-MP should be straightforward.

### 1.3 Hardware Platforms Supported by XPRESS-MP

**Console XPRESS** is available to run on a range of hardware, including PCs and UNIX workstations. **Visual XPRESS** is available on Windows-based machines. A model can be developed on one hardware platform and then transferred, usually without alteration, to another platform.

### 1.4 Symbols and Conventions

We have used the following conventions within this guide:

- Square brackets [...] contain optional material.
- Curly brackets {...} contain optional material, one of which must be chosen.
- Entities in *italics* which appear in expressions stand for meta-variables. The description following the meta-variable always describes how it is to be used.
- The ↵ symbol means "press the RETURN or Enter key on your keyboard". On a PC this is usually the big key marked ↵.
- The symbol CTRL followed by a letter means "hold down the Ctrl key and type the letter". The Ctrl or Control key is usually at the bottom left hand corner of the keyboard.

- The vertical bar symbol | is found on many keyboards as |, but often confusingly displays on-screen without the small gap in the middle. In the UNIX world it is referred to as the pipe symbol. Note that this symbol is not the same as the character sometimes used to draw boxes on a PC screen. In ASCII, the | symbol is 7C in hexadecimal, 124 decimal.
- Examples of commands, models and their output are printed in a *Courier* font. Although XPRESS-MP commands can be typed either in upper or lower case, we have used upper case throughout this User Guide. Filenames are given in lower case *Courier*.
- Mathematical objects are presented in *italics*.

## 1.5 Running XPRESS-MP



**Visual XPRESS** is an integrated system, having the usual Windows conventions. Please see its Quick Tour for details on how to run it. This User Guide is, however, relevant as it takes you through the modelling process and language, which is common to **Visual XPRESS** and **Console XPRESS**.



**Console XPRESS** has two main components:

- the *model builder*, called `mp-model`; and
- the *optimiser* `mp-opt`.

To run either the model builder or optimiser, simply type the module name `mp-model` or `mp-opt`, optionally followed by your problem name. If you do not provide a problem name you will be prompted for one before you go on. To try it and verify that your installation has been successful, type:

```
mp-model ↵
```

remembering that the ↵ symbol means "press the Enter key". You will see a sign-on heading telling you which version of `mp-model` you have installed, followed by a prompt for a problem name. Under a Windows command prompt the screen will look something like this:

```
C:\> mp-model
XPRESS-MP Extended Model Builder Release xx.yy
(c) Copyright Dash Associates 1984-zzzz
Enter problem name >
```

---

You will normally arrive at mp-model's command prompt, >, by entering a problem name and pressing ↵, but for now we leave the modeller by typing ↵. mp-model will show the message:

```
Default problem name is $$$$$$  
Type HELP for summary information
```

Now type:

```
QUIT
```

mp-model will respond with:

```
Model Specification time: xxx  Generation time:0
```

and return you to the command prompt.

You can check the successful operation of the optimiser in a similar way. Run the optimiser mp-opt, give it a blank line to satisfy its need for a problem name, and then immediately leave the program by quitting.

```
C:\> mp-opt↵  
XPRESS-MP Extended Integer Optimiser Release xx.yy  
(c) Copyright Dash Associates 1984-zzzz  
Enter problem name >↵  
Default problem name of $$$$$$ assumed  
>QUIT
```

If you see the banner headings for the modeller and optimiser then you can be fairly sure that the software has been installed correctly. If something goes wrong, please read the Release Notes again carefully, and re-start. If there still appears to be a problem, try again, talking through what you are doing with a colleague. If you continue to have problems contact your supplier.

## 1.6 The Structure of this Guide

**Chapter 2** of this Guide takes you through a simple modelling exercise and shows you how to convert the model you produce on paper to the form expected by XPRESS-MP, and then how to solve the model.

**Chapter 3** follows on with two further simple examples. We see how to use models that have been created in files, how to create tables to hold data, how to fill these tables with data from external sources, and how to create arrays of decision variables.

In **Chapter 4** we present a more challenging modelling problem in the form of a case study. It illustrates many of the features of XPRESS-MP, including the use of Integer Programming.

---

*Chapter 5* introduces some advanced features. We discuss how to generate variables and constraints only if some conditions are true. Further details are given of advanced methods of importing data, and there is a description of the modeller's built-in functions.

If, after reading the first two chapters, you feel confident you have got an intuitive understanding of the modelling language, you might move straight on to *Chapter 4*, since it covers the same material as *Chapter 3* but at a faster pace. Alternatively, look at *Chapter 3*, and then skim through *Chapter 4*.

We also suggest you take a brief look through *Chapter 5*, noting any features which you feel would be of particular benefit to you later.

If you are interested in Integer Programming then you should read *Chapter 6*.

At the end of this guide you will find a *Glossary of Terms* and an *Index*.



---

## 2 Getting Started with XPRESS-MP

### 2.1 Entering a Model

In this chapter we will take you through a very small manufacturing example to illustrate the basic building blocks of XPRESS-MP.



Models can be typed directly into the **Console XPRESS** model builder, `mp-model`, or they can be submitted in the form of files. There are advantages in presenting the model and data in file form because they can be edited easily if an error occurs or you wish to modify the model in some way. If the model and data are entered directly the input cannot subsequently be corrected or modified. However, when you want to test out ideas on a small scale in a one-off manner it can be useful to be able to enter your problem description directly. The Chess Set problem below shows how a model can be entered directly into `mp-model` while Sections 3 and 4 describe how to enter models from text files.



In **Visual XPRESS** you use the built-in intelligent editor to enter a model. Models can be saved to files, and a built-in Problem Library stores all models and XPRESS-MP controls that you use, to make it easy to re-load them.

### 2.2 The Chess Set Problem: Description

A joinery makes two different sizes of boxwood chess sets. The smaller requires 3 hours of machining on a lathe and the larger only requires 2 hours, because it is less intricate. There are ten lathes with skilled operators who each work a 40 hour week. The smaller chess set requires 1 kg of boxwood and the larger set requires 3 kg. However boxwood is scarce and only 200 kg per week can be obtained.

When sold, each larger chess set yields a profit of £20 and each smaller chess set a profit of £5. The problem is to decide how many sets of each kind should be made each week to maximise profit.

---

### 2.2.1 A First Formulation

The joinery can *vary* the number of large and small chess sets produced: there are thus two *variables* in our model. We shall give these variables abbreviated names:

$x_s$ : the number of small chess sets to make

$x_l$ : the number of large chess sets to make

The number of large and small chess sets we should produce to achieve the maximum contribution to profit is determined by the optimisation process. Since the values of  $x_s$  and  $x_l$  are to be determined by optimisation these are known as the *decision variables*, or more simply the *variables*.

The values which  $x_s$  and  $x_l$  can take may be *constrained* to be equal to, less than or greater than some constant. The joinery has a maximum of 400 hours of machine time available per week. Three hours are needed to produce each small chess set and two hours are needed to produce each large set. So the number of hours of machine time actually used each week is  $3x_s + 2x_l$ . One *constraint* is thus:

$$3x_s + 2x_l \leq 400 \quad (\text{hours})$$

This restricts the allowable combinations of small and large chess sets to those that do not exceed the man-hours available. In addition, only 200 kg of boxwood is available each week. Since small sets use 1 kg for every set made, against 3 kg needed to make a large set, a second constraint is:

$$x_s + 3x_l \leq 200 \quad (\text{kg})$$

The joinery cannot produce a negative number of chess sets so two further constraints are:

$$x_s \geq 0$$

$$x_l \geq 0.$$

The *objective function* is a linear function which is to be optimised, that is, maximised or minimised. It will involve some or all of the decision variables. In maximisation problems the objective function usually represents profit, turnover, output, sales, market share, employment levels or other "good things". In minimisation problems the objective function describes things like total costs, disruption to services due to breakdowns, or other less desirable process outcomes.

The aim of the joinery is to maximise profit. Since a large set contributes £20 to total profit while a small set contributes £5, the objective function is:

$$\text{profit} = 5x_s + 20x_l.$$

The collection of variables, constraints and objective function that define our linear programming problem is called a *model*.

---

## 2.3 Solving the Chess Set Problem

### 2.3.1 Building the model

The Chess Set problem can be solved easily using XPRESS-MP. The first stage is to get the model we have just developed into the XPRESS-MP language. The model has three sections: in the VARIABLES section the decision variables are specified, one per line; in the CONSTRAINTS section, the constraints and objective function are specified, one per line; and in the last section the GENERATE command tells XPRESS-MP that we have finished specifying the model.

Remember that we use the notation that items in italics (for example,  $x_s$ ) are the mathematical variables. The corresponding XPRESS-MP variables will be the same name in non-italic courier (for example, `xs`).

Once the model has been defined, the objective function can be optimised and the optimal number of large and small chess sets obtained.



If you are using **Console XPRESS** begin by calling `mp-model`. At the system prompt type:

```
mp-model↵
```

At the `Enter problem name>` prompt type `chess` and press ↵ to give the problem a name:

```
Enter problem name> chess↵
```



Start up **Visual XPRESS**. Select **File > New > Problem** which will bring up the New Problem dialog. Type `chess` in the Problem Name box. It is possible to enter a short description of the problem in the Problem Description box, but do not enter a Problem File. Click OK.

Note that if a problem called `chess` already exists you will have to choose another name such as `chess1`.

To identify the decision variables to XPRESS-MP, enter the *keyword* VARIABLES followed by the names of the decision variables, one per line.

```
VARIABLES
xs
xl
```

---

The second and third lines are indented purely for clarity. Any information on a new line following the VARIABLES keyword is treated as a variable definition until the next keyword is read.

The constraints and objective function are now described to XPRESS-MP, which requires that each constraint be given a name. Suggested constraint names are `mc_time` for machine time available, and `wood` for the amount of wood obtainable. To enter these constraints, the keyword `CONSTRAINTS` must be typed and each constraint specified. The constraint name must be followed by a colon and then by the inequality:

```
CONSTRAINTS
  mc_time: 3*xs + 2*xl < 400
  wood:    xs + 3*xl < 200
```

The `CONSTRAINTS` section requires some explanation. A constraint is made up from:

a constraint name,	followed by
a colon (:),	followed by
a linear expression,	followed by
a constraint type,	usually followed by
a right-hand-side value.	

A *linear expression* is a set of terms of the form:

$\pm \text{constant} * \text{variable}$

In practice, terms can occur on either side of the constraint type, but this is just a syntactic convenience.

*Constraint types* are:

< or <=	meaning a $\leq$ constraint;
> or >=	meaning a $\geq$ constraint;
=	meaning an = constraint; or
\$	meaning an objective function (an unconstrained row).

Notice that the character "\*" is used to denote multiplication of the decision variables by the units of machine time and wood that one unit of each uses in the `mc_time` and `wood` constraints, and that the symbol "<" is used to mean *less than or equal to* - there is no  $\leq$  sign on the standard keyboard. Blanks are not significant and by default the modelling language distinguishes between upper and lower case, so `X1` would be recognised as different from `x1`.

The objective function is defined as a constraint. The special character \$ is used at the end of the linear expression to inform XPRESS-MP that this "constraint" is the objective function. No value should be given after the \$ symbol. Enter the objective function thus:

---

```
profit: 5*xs + 20*xl $
```

By default, XPress-MP assumes that all variables are constrained to be non-negative unless it is informed otherwise. There is therefore no need to specify non-negativity constraints on variables. (Refer to the **BOUNDS Section** in chapter 4, *The mp-model Model Builder*, of the *XPress-MP Reference Manual* for further details.)

To conclude the model specification, we have:

```
GENERATE
```

In **Console XPress** this instructs mp-model to generate a representation of the problem suitable for optimisation by mp-opt. In **Visual XPress** it tells the system that you have finished specifying the model.

The complete model should look like this:

```
VARIABLES
  xs
  xl
CONSTRAINTS
  mc_time:  3*xs + 2*xl < 400
  wood:      xs + 3*xl < 200
  profit:    5*xs +20*xl $
GENERATE
```



If you are entering the model straight into mp-model then it is read as you type it in, checked for errors and a file is generated, known as the *matrix* (or *MPS*) *file*, called chess.mat. The problem name is chess.

mp-model will inform you of any errors and give some indication of what the error might be. In most cases it will ignore the erroneous line, but once a line has been accepted it cannot be retrieved for correction. If errors were detected in the input, then when you type the GENERATE command mp-model will ask if you want to create the matrix file. At this point you may prefer to type QUIT to exit without doing so.



In **Visual XPress** you enter the entire model before starting to run it. The syntax is not checked line by line. Any errors will be signalled when you try to solve the problem.

We shall assume that you made no errors in the model for now. Identifying and correcting errors is discussed in the next chapter.

It should be remembered that the XPress-MP modeller is case sensitive, i.e., SUPPLY and supply will be treated as two entirely different variables. Only for keywords

---

(INDICES, VARIABLES, DATA, CONSTRAINTS, GENERATE etc.) can upper and lower case be interchanged.

### 2.3.2 Obtaining a Solution using XPRESS-MP



Generating the MPS file should return you to the operating system prompt. Assuming no input errors, the problem can now be solved using the **Console XPRESS** optimiser, mp-opt. Start mp-opt by typing:

```
mp-opt
```

At the Enter problem name> prompt, type chess to tell it to use the problem named chess:

```
Enter problem name> chess
```

To read in the contents of the matrix file chess.mat, type:

```
INPUT
```

mp-opt gives a log of its progress and should respond with:

```
Reading Problem CHESS
Problem Statistics

          3 (          0 spare) rows
          2 (          0 spare) structural columns
          6 (          0 spare) non-zero elements
Global Statistics
          0 entities          0 sets          0 set members
```

To maximise the objective function type:

```
MAXIMISE
```

This should produce the following technical information that can be safely ignored for the moment:

```
Crash basis containing      1 structural columns created
  Its      Obj Value      S  Ninf  Nneg      Sum Inf  Time
    0      666.666667      p    0    0      .000000    0
    2     1333.333333      P    0    0      .000000    0
Uncrunching matrix
    2     1333.333333      P    0    0      .000000    1
Optimal solution found
```

The final line tells us that a solution has been found. The line before this gives the optimal value of the objective function, 1333.33333.



When you have entered the model, open Problem Optimiser Options and change the Sense to MAX, so the problem will be maximised. Then clicking Run Solve LP will parse the model and go on to solve it if there are no errors. If there are any errors in the model, these will be listed in a window at the bottom of the screen. You can correct any errors in the model and resolve the problem, but we assume there are no errors for now.

### 2.3.3 Interpreting the Output



To see the solution one screen at a time type:

PRINT

You will see a display which should look like that below. Data in boxes will be referred to later.

```
Problem Statistics
Matrix CHES
Objective profit
RHS RHS00001
Problem has      3 rows and      2 structural columns
```

```
Solution Statistics
Maximisation performed
Optimal solution found after      1 iterations
Objective function value is 1333.333374
```

type c/r to continue, anything else to finish >

Rows Section

Number	Row	At Value	Slack Value	Dual Value	RHS
N 1	profit__	BS 1333.333333	-1333.333333	.000000	.000000
L 2	mc_time_	BS 133.333328	266.666656	.000000	400.000000
L 3	wood____	UL 200.000000	.000000	6.666666	200.000000

type c/r to continue, anything else to finish >

Columns Section

Number	Column	At Value	Input Cost	Reduced Cost
C 4	xs_____	LL .000000	5.000000	1.666667
C 5	xl_____	BS 66.666664	20.000000	.000000

If you wish to inspect the solution at your leisure the PRINT command can be replaced by FPRINT (Full PRINT) which produces a file chess.prt containing the full solution. This file can be printed off or inspected using an editor or word

---

processor. Notice that trailing underscore characters have been added to the constraint and variable names to pad them out to 8 characters.

The sequence of commands is therefore:

```
mp-opt
chess
INPUT
MAXIMISE
FPRINT
```

Note that the problem name can be specified in the command used to start mp-opt, for example: mp-opt chess. This can also be done when calling mp-model.

Finally, typing:

```
QUIT
```

gets you out of mp-opt and back to the operating system prompt.



After successfully solving the problem, you can see a solution display by clicking Run Solution Print File. This writes the solution to a file and gives you the option to view it. You will see a display like the one above.

The names of the decision variables and their optimal values have been highlighted in the display above. The first line in the Columns Section shows that the optimum value of variable  $x_s$  is 0, i.e., the joinery should make no small chess sets. The optimum value obtained for variable  $x_l$  is 66.66, i.e., the joinery should make  $66\frac{2}{3}$  large chess sets (but no small chess sets) in order to maximise profit.

XPRESS-MP also shows the optimal profit produced. This can be found in the Rows Section above. The optimal profit from making  $66\frac{2}{3}$  large chess sets, each giving 20 units of profit, is  $1333\frac{1}{3}$ . Also displayed in the Rows Section is the amount of machine time (mc\_time) used, in this case  $133\frac{1}{3}$  units, and the amount of wood (wood) used, which is 200 kg.

In the present example the solution is not a whole number: optimal profit is obtained from making  $66\frac{2}{3}$  chess sets and not 67, which would require more wood than is available, or 66, which would leave some spare manufacturing capacity. In reality this may be practical since a chess set that is not completed in one week's production schedule may be completed the following week. However, some optimisation problems have variables that cannot be fractional in this way. Such problems require the techniques of Mixed Integer Programming. Further details of this approach can be found in **Chapter 6, Integer Programming** in the *XPRESS-MP User Guide*.

Although we have found an optimal solution to the problem given, this may not always be possible. If the constraints were insufficient it may be possible to improve the objective function indefinitely. In this case, the problem is said to be *unbounded*. This

---

usually means that a relevant constraint has been omitted from the problem or that the data are in error, rather than indicating an infinite source of profit. Conversely, if the constraints are too restrictive it may not be possible to find any values for the decision variables that satisfy all constraints. In this case, the problem is said to be *infeasible*. This would be the case in the chess set problem if an order of 100 large chess sets had to be made each week.

## 2.4 Summary

By now you should be able to

- Build a simple model.



- Recognise that **Console XPRESS** has two components, `mp-model` and `mp-opt` linked by a matrix file. Both can be accessed from the operating system prompt.
- Put the model into the format expected by `mp-model`.
- Run `mp-model`.
- Solve the problem with `mp-opt`.
- Display the optimal solution values.
- Obtain a print-out of the solution.



- Enter a model and solve it.
- Obtain a print-out of the solution.

In the next chapter two further models will be considered. They are a little more complex and will give you a chance to become familiar with some of the more powerful features of the XPRESS-MP modelling language.



---

## 3 Two Simple Examples

This chapter develops the basics of modelling set out in Chapter 2. It presents two further examples of the use of XPRESS-MP and introduces two new features.



The first of these is the use of *model files*. Model files are text files which contain a description of the model in `mp-model` format created using a standard text editor. Presenting models to `mp-model` from such files saves having to re-type a model when it is changed and is fundamental to any model management scheme.



**Visual XPRESS** "knows about" its own model files, and has its own editor. **Visual XPRESS** also stores problems as files. These *problem files* include the model description, equivalent to the model file in `mp-model`, and the settings of the optimiser controls, such as whether the objective is to be minimised or maximised. **Visual XPRESS** has its own editor to develop and maintain problems.

Example 1 shows how model files are used with XPRESS-MP.

The second new feature is the use of data tables. These contain the data to be used with a model. Data for these tables can be entered from a variety of sources such as ASCII text files and Excel spreadsheets. In the second example, a more advanced model is described which uses data input from ASCII files.

### 3.1 Example 1: A Simple Production Planning Exercise

In this example we build up a model piece by piece in preparation for saving it to a model file (a `.mod` or `.wmd` file in XPRESS-MP conventions). We also illustrate the use of data tables and how to enter data directly into these tables from within the modeller.

---

### 3.1.1 The Model Background

A firm is processing three products A, B and C on two machines  $M_1$  and  $M_2$ . Each day there are 8 hours available on  $M_1$ , and 10 on  $M_2$ . The machine hours required per 100 units of the products are given in the following table:

	A	B	C
$M_1$	2.2	1.8	1.9
$M_2$	2.4	2.0	2.1

Currently the profit contributions per 100 units of the products are:

A	24.7
B	22.4
C	19.7

Letting  $a$  represent the number (in hundreds) of units of product A to be made, and  $b$  and  $c$  similarly, our LP is:

$$\begin{aligned} \text{Maximise PROFIT} &= 24.7a + 22.4b + 19.7c \\ \text{Subject to:} & 2.2a + 1.8b + 1.9c \leq 8 & (M1) \\ & 2.4a + 2.0b + 2.1c \leq 10 & (M2) \end{aligned}$$

and  $a, b, c$  are non-negative.

### 3.1.2 Developing the Model

We required three decision variables  $a, b$  and  $c$  to represent the quantity of the products A, B and C respectively to produce. The following input to XPRESS-MP will define the variables required.

```
VARIABLES
a
b
c
```

Now we define two *data tables*:  $REQ(2, 3)$  to store the machine time requirements for each product-machine combination and  $PC(3)$  to store the profit contributions from each product. These data tables are just like arrays in mathematics. When we define  $REQ(2, 3)$  in a TABLES section we create a 2-dimensional array with 2 rows and 3 columns. Defining  $PC(3)$  we create a 1-dimensional array with 3 entries. If we wish to refer to the entry in the second row and first column of  $REQ(2, 3)$  then we specify  $REQ(2, 1)$ .

---

In the modeller we create tables, which initially have all zero entries, by the lines:

```
TABLES
REQ(2,3)
PC(3)
```

We can put some *data* into these tables using the simplest of the methods available. More flexible and powerful techniques will be considered later. First the machine requirements. We can enter these row-by-row as:

```
DATA
REQ(1,1)= 2.2, 1.8, 1.9
REQ(2,1)= 2.4, 2.0, 2.1
```

The DATA section lets you specify a starting position in a table, and then a stream of numbers which will be put into the table starting at the specified position and carrying on into cells to the right. In the second and third lines above,

```
REQ(1,1) gets the value 2.2,
REQ(1,2) gets the value 1.8,
REQ(1,3) gets the value 1.9;
REQ(2,1) gets the value 2.4,
REQ(2,2) gets the value 2.0, and finally
REQ(2,3) gets the value 2.1.
```

Similarly, for the profit contributions table. To initialise this we have:

```
PC = 24.7, 22.4, 19.7
```

If the coefficient of the starting position in a table is not specified it is assumed to be the first row and column in the table. Further examples can be found in **DATA Section** in chapter 4, *the mp-model Model Builder*, of the *XPRESS-MP Reference Manual*.

We have now defined the decision variables whose values are to be determined by the optimisation process and seen the commands to enter the required data into XPRESS-MP tables. The constraints can now be defined as follows:

```
CONSTRAINTS
PROFIT:PC(1)*a +PC(2)*b +PC(3)*c $
M1: REQ(1,1)*a +REQ(1,2)*b +REQ(1,3)*c < 8
M2: REQ(2,1)*a +REQ(2,2)*b +REQ(2,3)*c < 10
```

We will terminate the model by the line:

```
GENERATE
```



Models are usually assembled in files with an extension of .mod, where the first part of the filename can be selected by the user to reflect the problem being modelled.

---

You should now assemble all the input items together, in the order that we have developed them, and enter the text into a file `prod.mod`.

If you are using a word-processor to create the model remember to use it in "non-document" mode. The file must consist of ASCII characters only, with no embedded control characters.



In **Visual XPRESS** you just have to enter the complete model, using the built-in editor.

Here is the complete model:

```
VARIABLES
  a
  b
  c
TABLES
  REQ(2,3)
  PC(3)
DATA
  REQ(1,1) = 2.2, 1.8, 1.9
  REQ(2,1) = 2.4, 2.0, 2.1
  PC       = 24.7, 22.4, 19.7
CONSTRAINTS
  PROFIT: PC(1)*a +PC(2)*b +PC(3)*c $
  M1: REQ(1,1)*a +REQ(1,2)*b +REQ(1,3)*c < 8
  M2: REQ(2,1)*a +REQ(2,2)*b +REQ(2,3)*c < 10
GENERATE
```

### 3.1.3 Correcting Syntax Errors in the Model



We can start `mp-model` by typing, at the operating system prompt:

```
mp-model prod
INPUT
```

The model will be read from the file `prod.mod`. `mp-model` will then create a matrix file `prod.mat`. If you made any errors typing in the model, `mp-model` will display an error message and ask you whether you wish to generate the model. Say no, type `QUIT` and return to your `prod.mod` file to correct any errors and resubmit the model to `mp-model`.

If the model is long or several errors are reported, you can review all of the error messages in context by examining the file `prod.lst`. This *list* (`.lst`) file contains

---

a log of all output produced by `mp-model`, including the model statements, error messages and the output of any `PRINT` commands.

Each error message is preceded by a question mark '?' which makes it easy to search for them in the list file – provided you haven't included any question marks in comments in your model! Following the question mark comes the error number, some numbers in parentheses, and finally an error message. The error message is often sufficient to diagnose the error, but if not, a more detailed explanation and possible causes and remedies are given in the text file `moderror.txt`. This file lists the errors by error number and is included in the XPress installation directory.

The numbers in parentheses identify the line number at which the error was detected. The first number is the line number at the first level of input, and the second number if given represents the line number at the second level of input. So if you run `mp-model` as follows

```
mp-model prod
INPUT
```

and `mp-model` generates the error message

```
?26(1.53):name not found: sx
```

it means that error number 26 occurred on line 53 of the file `input` at the first line typed into `mp-model`.

If you are using the `.lst` file to review errors, remember to make corrections to the `.mod` file – correcting the `.lst` file is a common mistake!

Often one error will result in many errors further on in the model. Instead of trying to correct all of the errors at once, you may find it easier to correct the initial errors and any further obvious errors and re-submit the model to `mp-model`. If any errors still remain, you can repeat the process until all errors have been removed.



If you make any mistakes in a model in **Visual XPress**, they will be detected when you try to solve the problem and listed in a dialog box at the bottom of the **Visual XPress** window. The main panel of the dialog lists all of the errors giving an error number, a line number in parentheses, and an error message in the same format as **Console XPress** error messages described above.

If you double click on one of the error messages, the cursor in the model will move to the line on which the error was detected. You can then correct the error and immediately re-solve the problem if you like, to check that you have corrected the error. Often one error will result in several more errors further on in the model so it is a good idea to work through the errors in sequence, re-solving the problem each time.

---

If you need further advice on any of the error messages, look the error number up in the text file `moderror.txt`. It gives a more detailed explanation of each error together with possible causes and remedies.

One final word of warning – the fact that there are no syntax errors in the model doesn't mean that the model is correct! Always check that the solution you get makes sense.

### 3.1.4 Obtaining the Solution



Once an error-free model file has been submitted to `mp-model`, we can run the optimiser to obtain a solution. This is done by typing:

```
mp-opt prod
```

followed by:

```
INPUT
```

to read the MPS matrix `prod.mat`. To find the maximum profit, type:

```
MAXIMISE
```

`mp-opt` will produce some lines of output in the form of an iteration log; these can be ignored for the moment. However, if the model has been successfully optimised the last line of this output should say:

```
Optimal solution found
```

To see the solution type:

```
PRINT
```

and a solution summary will appear. Repeatedly hitting the `↓` key will go through the solution section by section. The solution summary can be printed to a text file `prod.prt` by using the `FPRINT` command. It will then be available for more leisurely inspection.

You can then leave the optimiser by typing:

```
QUIT
```



In **Visual XPRESS** you just have to change the optimisation direction to `MAX`, click `Run Solve LP`, and then `Run Solution Print File` to see the solution.

## 3.2 Example 2: A Blending Example

This example illustrates how data may be read into tables from ASCII text files. It also shows you how we can create *arrays of variables*, i.e., *subscripted variables*.

---

### 3.2.1 The Model Background

A mining company has two types of ore available: Ore 1 and Ore 2. Denote the amounts of these ores to be used by  $x_1$  and  $x_2$ . The ores can be mixed in varying proportions to produce a final product of varying quality. For the product we are interested in, the "grade" (a measure of quality) of the final product must be between the specified limits of 4.0 and 5.0. It sells for £125 per ton. The costs of the two ores vary, as do their availabilities.

Maximising net profit (i.e., sales revenue less cost of raw material) gives us the objective function:

$$NET\_PROFIT = \sum_{j=1}^2 (125 - COST_j) x_j$$

We then have to ensure that the grade of the final ore is within certain limits. Assuming the grades of the ores combine linearly, the grade of the final product is:

$$\frac{\sum_{j=1}^2 GRADE_j x_j}{\sum_{j=1}^2 x_j}$$

This must be greater than or equal to 4.0 so, cross-multiplying and collecting terms, we have the constraint:

$$\sum_{j=1}^2 (GRADE_j - 4.0) x_j \geq 0$$

Similarly the grade must not exceed 5.0, so we have the further constraint:

$$\sum_{j=1}^2 (5.0 - GRADE_j) x_j \geq 0$$

Finally there is a limit to the availability of each of the ores. We model this with the constraints:

$$\begin{aligned} x_1 &\leq AVAIL_1 \\ x_2 &\leq AVAIL_2 \end{aligned}$$

### 3.2.2 Developing the Model

The above problem description sets out the relationships which exist between variables but contains few explicit numbers. Focusing on relationships rather than figures makes

---

the model much more flexible. In this example only the selling price and the upper/lower limits on the grade of the final product are fixed.

In developing the model we assume that the grades of the ores are available in an ASCII data file called `quality.dat`. Using an editor (or **Visual XPRESS**), create a data file called `quality.dat` and insert a single line representing the grades of ores 1 and 2.

```
2.1, 6.3
```

We will also assume that the varying costs and availabilities are in ASCII files, `cost.dat` and `avail.dat`.

Into file `cost.dat` put

```
85.0, 93.0
```

and into `avail.dat` put

```
60, 45
```

We have specified 60 tons of ore 1 and 45 tons of ore 2 but you may wish to experiment with different values.

The next step is to rewrite the algebraic model we began with in a format which the XPRESS-MP modeller can interpret. Note that comments can be included if prefixed with an exclamation mark "!".

Enter the following model, and call it `blend2`. Note that if you are using `mp-model` you will put this in a file `blend2.mod`, whilst in **Visual XPRESS** you will create a problem called `blend2`, saved to the file `blend2.wmd`.

```
VARIABLES
    x(2)                ! Quantity of each ore to purchase

TABLES
    COST(2)             ! Cost per ton of ores
    AVAIL(2)            ! Availability of ores
    GRADE(2)            ! Quality of ores

DISKDATA
    GRADE = quality.dat    ! Input grades from file
    COST  = cost.dat       ! Costs
    AVAIL = avail.dat      ! Availabilities

CONSTRAINTS
    ! Maximise (Revenue-Costs)
    max: SUM(j=1:2) (125-COST(j))*x(j) $

    ! Grade < Upper limit
```

---

```

QualMax: SUM(j=1:2) (5.0-GRADE(j))*x(j) > 0

! Grade > Lower limit
QualMin: SUM(j=1:2) (GRADE(j)-4.0)*x(j) > 0

max_x1:   x(1) < AVAIL(1)      ! Limits on ore
max_x2:   x(2) < AVAIL(2)      ! availabilities

```

```
GENERATE
```

In this model file, the two ore variables have not been identified uniquely by name. Instead we have defined an array, where the variables can be found. This makes the model more general and means that it can be applied to problems involving many types of ore if required. The array is written:

```
VARIABLES
  x(2)
```

which creates a one-dimensional array of variables,  $x(1)$  and  $x(2)$ . An array written as:

```
VARIABLES
  y(5,8)
```

would create a 5 row by 8 column array of  $y$  variables. The variable in the second row, third column would be referred to as  $y(2,3)$ .

When data are input from a file then XPRESS-MP must be told the filename. This is done using the DISKDATA section:

```
DISKDATA
  GRADE = quality.dat      ! Input grades from file
```

specifies that the table GRADE is to be filled with data from the text file quality.dat. DISKDATA replaces the DATA section when data are obtained from a file rather than entered directly.



Use an editor to enter the model above into a file blend2.mod.

Having set up the data files as required we can call up mp-model and read in the model using the commands:

```
mp-model blend2
INPUT
```

The ore costs, availabilities and qualities will be extracted from the various input files as necessary by mp-model. A matrix file blend2.mat will be produced, ready for input into the optimiser.

---

### 3.2.3 Obtaining the Solution



We can now run the optimiser, using:

```
mp-opt blend2
```

Type:

```
INPUT
```

to read the MPS matrix. To find the maximum net profit, type:

```
MAXIMISE
```

and you will see an iteration log. To see the solution, enter:

```
PRINT
```

and a solution summary will appear. Repeatedly hitting the ↵ key will go through the solution section by section. You can then leave the optimiser by typing:

```
QUIT
```



In **Visual XPRESS**, after selecting the direction of optimisation, you just have to click Run Solve LP, and then Run Solution Print File to see the solution.

Note that the  $x$  variables  $x(1)$  and  $x(2)$  will appear in the solution listing as "x\_\_\_\_\_01" and "x\_\_\_\_\_02". These 8 character names are known as *MPS names* and are the names used in the matrix file.

### 3.2.4 Re-running the Model with New Data

Now suppose you want to run the model again with different costs and availabilities for the two ores. Simply edit the .dat files and make the required changes.

XPRESS-MP will extract the revised data from the files. It will then generate a new matrix file and you can go on to optimise the new model, just as before.

By now you should be able to see the advantages of storing the model either in a text file for **Console XPRESS**, or in the problem database for **Visual XPRESS** - there is no need to type in the model each time you wish to run it. So long as the model's *structure* does not change, the model can be used again and again to obtain a new optimal solution whenever the data producing a particular *model instance* change.

---

## 4 The Coco Problem

### 4.1 Overview

This chapter shows how XPRESS-MP can be used to solve a one-period optimisation problem in manufacturing which develops into a multi-period problem. It also illustrates some aspects of integer programming. The problem splits into five phases. The first phase shows how the basic manufacturing problem is formulated. The second phase elaborates on the use of arrays first introduced in Chapter 3. Multi-period modelling is introduced in phase 3 while phase 4 deals with the problem of fixed costs. In the fifth and final phase, we see how XPRESS-MP's integer programming facility allows us to determine what benefits are to be gained if a particular factory should be closed permanently rather than remain open.

The problem formulations remain simple, but the structure of the problem shows how XPRESS-MP's powerful notation makes building even complicated models an easy task.

### 4.2 Problem Description

The Coco company is faced with the problem of planning its production for the next quarter. Coco has two factories ( $F_1$  and  $F_2$ ), each capable of manufacturing two products ( $P_1$  and  $P_2$ ). The selling prices of the products in the next quarter have been estimated by the marketing department to be (in \$/tonne):

$P_1$	400	Table <i>SP</i>	(Selling Price)
$P_2$	410		

Each product is made from two raw materials,  $R_1$  and  $R_2$ , whose prices in the next quarter are expected to be (in \$/tonne):

$R_1$	100	Table <i>RMP</i>	(Raw Material Price)
$R_2$	200		

The raw materials required for each tonne of product are as follows:

	$R_1$	$R_2$	Table <i>RMREQ</i>	(Raw Material Required)
$P_1$	1.0	0.5		
$P_2$	1.3	0.4		

e.g., 1 tonne of product  $P_2$  requires 1.3 tonnes of material  $R_1$  and 0.4 tonnes of material  $R_2$ .

The factories vary in their manufacturing efficiencies and the variable costs of manufacture are (in \$/tonne):

	$F_1$	$F_2$		
$P_1$	150	153	Table VC	(Variable Costs)
$P_2$	75	68		

This shows that Coco incurs a cost of \$150 for each tonne of product  $P_1$  made at factory  $F_1$ , in addition to the cost of the raw materials used to manufacture product  $P_1$ .

The maximum total amount of product that can be made at each factory is (in tonnes):

	$F_1$	$F_2$		
	400	500	Table MAXPROD	(Maximum Production)

and the marketing department has estimated that maximum sales levels for the two products in the next quarter will be (in tonnes):

	$P_1$	$P_2$		
	650	600	Table MAXSELL	(Maximum Selling Level)

### 4.3 Phase 1: A First Formulation

#### 4.3.1 Specifying the Variables

Coco wants to know how many tonnes of each product to make at each factory so as to achieve the maximum profit contribution. These decisions are the *variables* of the model, and must be declared in the VARIABLES section of the model.

We shall give them meaningful names. Let's call them:

- $manuf_{11}$ , for the amount of product  $P_1$  manufactured at factory  $F_1$
- $manuf_{12}$ , for the amount of product  $P_1$  manufactured at factory  $F_2$
- $manuf_{21}$ , for the amount of product  $P_2$  manufactured at factory  $F_1$
- $manuf_{22}$ , for the amount of product  $P_2$  manufactured at factory  $F_2$

As is usual in LP, these variables must be non-negative. XPRESS-MP will assume non-negativity unless it is explicitly told otherwise.

#### 4.3.2 Specifying the Constraints

The next section of the model deals with the constraints imposed on variables individually and in combination, and sets out the objective function. Coco has very simple constraints, but a relatively complicated objective function. We will deal with the constraints first.

---

Coco cannot produce more than 400 tonnes in total at factory  $F_1$  (see Table *MAXPROD*). This gives us the constraint:

$$manuf_{11} + manuf_{21} \leq 400 \quad (MAXATF1)$$

i.e., the amount of  $P_1$  made at  $F_1$  plus the amount of  $P_2$  made at  $F_1$  is less than or equal to 400 tonnes. Similarly Coco cannot produce more than 500 tonnes in total at factory  $F_2$  which gives the constraint:

$$manuf_{12} + manuf_{22} \leq 500 \quad (MAXATF2)$$

XPRESS-MP requires each constraint to have a name. The first constraint has been labelled *MAXATF1*, the second *MAXATF2*.

There are two constraints on the maximum amount of each product that Coco can sell (see Table *MAXSELL*). For product  $P_1$

$$manuf_{11} + manuf_{12} \leq 650 \quad (MAXMK1)$$

i.e., the amount of  $P_1$  made at  $F_1$  plus the amount of  $P_1$  made at  $F_2$  must be less than or equal to the maximum sales level expected; it would be wasteful to over-produce. Likewise, for product  $P_2$

$$manuf_{21} + manuf_{22} \leq 600 \quad (MAXMK2)$$

The objective function is a special form of constraint. In Coco's case, the objective function is not straightforward and needs to be built up carefully. Let us first consider selling product  $P_1$ . If we sell 1 tonne we get a revenue of \$400 (looking at Table *SP*) and incur raw material costs of  $1.00 \times \$100$  from raw material  $R_1$  plus  $0.50 \times \$200$  from raw material  $R_2$  (see Tables *RMREQ* and *RMP*). The total raw material cost is thus \$200 per tonne of  $P_1$  produced (i.e.,  $1.0 \times \$100 + 0.5 \times \$200 = \$200$ ).

If product  $P_1$  is manufactured at factory  $F_1$ , then the contribution to profit is \$400 - \$200 - \$150, given by the revenue minus the cost of raw materials and the variable production cost associated with manufacturing  $P_1$  at factory  $F_1$  (see Table *VC*). The objective function coefficient for  $manuf_{11}$  is therefore  $(400-200-150)=50$ .

The same procedure must be used to obtain an objective function coefficient for the three remaining variables. When all the coefficients have been found the complete objective function can be specified as follows:

$$50manuf_{11} + 47manuf_{12} + 125manuf_{21} + 132manuf_{22} \quad (PROFITC)$$

This is the function that Coco wishes to maximise.

---

To recap, our aim is to maximise

$$50\text{manuf}_{11} + 47\text{manuf}_{12} + 125\text{manuf}_{21} + 132\text{manuf}_{22} \quad (\text{PROFITC})$$

subject to:

$$\text{manuf}_{11} + \text{manuf}_{21} \leq 400 \quad (\text{MAXATF1})$$

$$\text{manuf}_{12} + \text{manuf}_{22} \leq 500 \quad (\text{MAXATF2})$$

$$\text{manuf}_{11} + \text{manuf}_{12} \leq 650 \quad (\text{MAXMK1})$$

$$\text{manuf}_{21} + \text{manuf}_{22} \leq 600 \quad (\text{MAXMK2})$$

and the non-negativity constraints:

$$\text{manuf}_{11} \geq 0, \text{manuf}_{12} \geq 0, \text{manuf}_{21} \geq 0, \text{manuf}_{22} \geq 0$$

#### 4.3.3 Using XPRESS-MP to Solve the Coco Problem.

The Coco problem can easily be solved using XPRESS-MP in the usual way. If **Console XPRESS** is being used, `mp-model` will take a representation of the model and turn it into the industry standard MPS matrix file; while `mp-opt` will read this matrix file and solve it to find the best values of the decision variables. In **Visual XPRESS**, enter the model, save it and solve it.

Create a model file `coco1` containing the following model statements:

```
VARIABLES          ! Introduce the variables
manuf11
manuf12
manuf21
manuf22

CONSTRAINTS        ! Define the objective function
                    ! and constraints

MAXATF1:manuf11 + manuf21 < 400
MAXATF2:manuf12 + manuf22 < 500
MAXMK1: manuf11 + manuf12 < 650
MAXMK2: manuf21 + manuf22 < 600

PROFITC:50*manuf11+47*manuf12+125*manuf21+132*manuf22 $
GENERATE ! End of the model
```

Now solve the model, remembering to maximise.

It is a good idea to check that the *Problem Statistics* reported by `mp-opt` or **Visual XPRESS** are in agreement with what we expect. In this case we see that we have 5 rows (constraints) and 4 structural columns (another name for decision variables). The 5 rows come from 4 constraints and the objective function. We do, indeed, have 4 decision variables.

There are 12 non-zeros in the constraints and objective function, i.e., 2 in each of the 4 constraints and 4 in the objective function. The *Global Statistics* tell you how many Integer Programming objects there are in the problem. We see that there are none at present although some will appear later in Coco's deliberations.

When you inspect the solution the number of interest is 93500.00000, the optimal objective function value in the Obj Value column. This shows the best profit that Coco can achieve. The other numbers may be of interest if you are concerned with the process that XPRESS-MP goes through to find the best solution.

#### 4.3.4 Viewing the Solution



If you want to find more about the policy that gives Coco this profit contribution, type PRINT: and get the following output. Hitting ↵ when prompted to "type c/r to continue, anything else to finish" will take you through the three sections of the solution.



In **Visual XPRESS**, click Run Solution Print File.

You should get the following output.

```
Problem Statistics
Matrix cocol
Objective PROFITC
RHS RHS00001
Problem has      5 rows and      4 structural columns
```

(Section 1)

```
Solution Statistics
Maximisation performed
Optimal solution found after      3 iterations
Objective function value is 93500.00000
```

type c/r to continue, anything else to finish >

```
Rows Section
Number  Row  At      Value  Slack Value  Dual Value  RHS
L      1  MAXATF1_ UL    400.00000    .000000    50.000000  400.000000
L      2  MAXATF2_ UL    500.00000    .000000    57.000000  500.000000
L      3  MAXMK1_   BS    300.00000   350.000000    .000000    650.000000
L      4  MAXMK2_   UL    600.00000    .000000    75.000000  600.000000
N      5  PROFITC_  BS   93500.00000 -93500.00000    .000000    .000000
```

type c/r to continue, anything else to finish >

(Section 2)

```
Columns Section
Number  Column At      Value  Input Cost  Reduced Cost
C      6  manuf11_ BS    300.000000   50.000000    .000000
C      7  manuf12_ LL      .000000   47.000000   10.000000
C      8  manuf21_ BS    100.000000  125.000000    .000000
C      9  manuf22_ BS    500.000000  132.000000    .000000
```

(Section 3)

---

The sections are most usefully considered in the order Section 1, Section 3 and then Section 2.

Section 1 contains summary statistics about the solution process and the optimal solution that has been found. It gives the matrix (problem) name (coco1) and the objective function and right-hand-side that have been used. Then follows the number of rows and columns, the fact that it was a maximisation problem, that it took 3 iterations (steps) to solve, and that the best solution has a value of 93500.

The optimal values of the decision variables are given in Section 3, the *Columns Section*. The *Number* is a sequence number. The name of the decision variable is given under the Column heading. Under *At* is the *status* of the column: BS means it is away from its lower or upper bound; LL means that it is at its lower bound; and UL (not seen here) means that the column is limited by its upper bound.

The *Value* column gives the optimal value of the variable. For instance, the best value for the variable *manuf11* is 300, and for variable *manuf21* it is 100. The *Input Cost* column tells you the coefficient of the variable in the objective function. Recall that *manuf11*'s per unit profit contribution was 50. (It is perhaps unfortunate that the industry standard heading refers to minimisation problems - properly the heading here for a maximisation problem should be "Input Profit").

Setting the contributions out in the table below makes it clear how the optimal profit of 93500 arises.

Product/Factory	Amount Made	Per Unit Contribution	Total Contribution
P1 at F1	300	50	15000
P1 at F2	0	47	0
P2 at F1	100	125	12500
P2 at F2	500	132	66000
TOTAL			93500

The final column in the solution print gives the *Reduced Cost* of the variable, which is always zero for variables that are away from their bounds - in this case, away from zero. Since Coco is not making any units of product P<sub>1</sub> at factory F<sub>2</sub> then we can assume that the per unit profit contribution is not high enough to make production viable. The reduced cost of *manuf12*, 10.0, shows that the per unit profitability of P<sub>1</sub> must increase by 10.0 before it would become worthwhile making any of the product at the factory. Alternatively, and this is where the name *reduced cost* comes from, the cost of manufacturing P<sub>1</sub> at F<sub>2</sub> would have to fall by 10.0 before Coco could make the product there without reducing its best profit.

The final section to consider is Section 2, the *Rows Section*.

Rows Section							
Number		Row	At	Value	Slack Value	Dual Value	RHS
L	1	MAXATF1_	UL	400.00000	.000000	50.000000	400.000000
L	2	MAXATF2_	UL	500.00000	.000000	57.000000	500.000000
L	3	MAXMK1_	BS	300.00000	350.000000	.000000	650.000000
L	4	MAXMK2_	UL	600.00000	.000000	75.000000	600.000000
N	5	PROFITC_	BS	93500.00000	-93500.000000	.000000	.000000

The first column shows the constraint type: L means a less-than-or-equal-to constraint; E indicates an equality constraint; G refers to a greater-than-or-equal-to constraint; and N means that it is an objective function.

The *sequence numbers* are in the next column, followed by the name of the constraint. The *At* column tells you the *status* of the constraint. A UL indicator says that the row is at its upper limit. In this case a  $\leq$  row is hard up against the right-hand-side that is constraining it. BS means that the constraint is not active and could be removed from the problem without changing the optimal value. If there were  $\geq$  constraints then we might see LL indicators, meaning that the constraint was at its lower limit. For example, if we had an expression which had to be greater-than-or-equal to some value then the optimal solution would meet the value at equality.

The *RHS* column is the right-hand-side of the original constraint, and the *slack* is the amount by which the constraint is away from its RHS. If we are tight up against a constraint (the status is UL or LL) then the slack will be 0. For instance, the slack on constraint MAXMK2 is 0. The constraint is:

$$\text{manuf21} + \text{manuf22} < 600$$

and we have seen that the values of  $\text{manuf}_{21}$  and  $\text{manuf}_{22}$  are 100 and 500 respectively. So the left-hand-side of the constraint (its value) is  $100+500=600$ , tight up against the right-hand-side. There is no "slack" on this constraint.

The *Dual Value* is a measure of how tightly a constraint is acting. If we are hard up against a  $\leq$  constraint then we expect to make more profit if the constraint is relaxed a little. The dual value gives a precise numerical measure to this intuitive feeling. In general terms, if the right-hand-side of a  $\leq$  row is increased by 1 then the profit will increase by the dual value of the row. More specifically, if the right-hand-side increases by a sufficiently small  $\delta$  then the profit will increase by  $\delta \times \text{dual value}$  since the dual value is a marginal concept. Dual Values are sometimes known as shadow prices.



If you would like to inspect the solution at your leisure use the sequence:

```
mp-opt cocol
INPUT
MAXIMISE
FPRINT
QUIT
```




The FPRINT (Full PRINT) command produces a file `cocol.prt` containing the full solution. This file can be printed off, or inspected using an editor or word processor.



As you may have realised, a **Run > Solution Print File** also creates a `cocol.prt` file containing the full solution. This file can be inspected and printed off directly in **Visual Xpress**.

#### 4.3.5 Where are We?

So far we have

- built a simple model
- got it into the format expected by Xpress-MP
-  run `mp-model`
-  solved the problem with `mp-opt`
-  solved it using **Visual Xpress**
- obtained a print-out of the solution
- understood the meaning of the solution

But there are many defects in our model and method. In its present form it is not easy to generalise the model to more than two products or two factories. The data, too, have been "hard-wired" in, and it would be tedious to change the model to reflect changes in the underlying data – we have not separated the model instance data from the model structure. In the next section we shall see how to overcome these limitations.

### 4.4 Phase 2: Using Subscripts

In the Xpress-MP modelling language we refer to the entry in the  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of table `TAB` as `TAB(i, j)`. So, for example, looking at the table *RMREQ* that we had earlier, *RMREQ*<sub>2,1</sub> is 1.3.

RMREQ	R <sub>1</sub>	R <sub>2</sub>
P <sub>1</sub>	1.0	0.5
P <sub>2</sub>	<b>1.3</b>	0.4

Similarly we can use arrays of decision variables and constraints so that problems can be formulated easily and flexibly. Nothing could be simpler. If  $manuf_{p,f}$  is defined to be the number of tonnes of product  $p$  manufactured at factory  $f$ , then the natural way to formulate our problem in mathematical notation would be:

$$\begin{aligned}
&\text{maximise} && \sum_{p=1}^2 \sum_{f=1}^2 PC_{p,f} manuf_{p,f} \\
&\text{subject to} && \sum_{f=1}^2 manuf_{p,f} \leq MAXSELL_p \quad \text{for } p = 1, 2 \\
&&& \sum_{p=1}^2 manuf_{p,f} \leq MAXPROD_f \quad \text{for } f = 1, 2 \\
&&& manuf_{p,f} \geq 0 \quad \text{for } p = 1, 2; f = 1, 2
\end{aligned}$$

where

$$PC_{p,f} = SP_{p,f} - \left( \sum_{r=1}^2 RMREQ_{p,r} RMP_r \right) - VC_{p,f}$$

We have now defined a new table  $PC_{p,f}$  (Profit Contribution) which is the profit contribution per unit of product  $p$  made in factory  $f$ .

In the language of XPRESS-MP we write:

```

VARIABLES          ! Introduce the variables
manuf(2,2)          ! A 2 by 2 array manuf(p,f)

TABLES              ! NOTE: there are no data in these
MAXPROD(2)          ! tables, so this model will not run
MAXSELL(2)
PC(2,2)

CONSTRAINTS
MAXMK(p=1:2): SUM(f=1:2)manuf(p,f) < MAXSELL(p)
MAXATF(f=1:2): SUM(p=1:2)manuf(p,f) < MAXPROD(f)
PROFITC:            SUM(p=1:2,f=1:2) PC(p,f)*manuf(p,f) $

END

```

Note: END is a synonym for GENERATE.

---

The constraint:

```
MAXATF(f=1:2): SUM(p=1:2)manuf(p,f) < MAXPROD(f)
```

corresponds to the mathematical constraint:

$$\sum_{p=1}^2 \text{manuf}_{p,f} \leq \text{MAXPROD}_f \quad \text{for } f = 1, 2$$

in the following manner:

```
MAXATF(f=1:2): says "generate a separate MAXATF constraint for all values of
the dummy subscript f ranging from 1 to 2".
```

```
SUM(p=1:2)manuf(p,f) says "sum, from p=1 to p=2, manuf(p,f) "
```

```
< MAXPROD(f) says "the upper bound on this constraint is MAXPROD(f) "
```

So the single line generates two constraints, the first for  $f=1$  and the second for  $f=2$ .

With  $f=1$  the constraint is:

```
MAXATF(1): SUM(p=1:2)manuf(p,1) < MAXPROD(1)
```

```
i.e., MAXATF(1): manuf(1,1) + manuf(2,1) < MAXPROD(1)
```

while for  $f=2$  it is

```
MAXATF(2): SUM(p=1:2)manuf(p,2) < MAXPROD(2)
```

```
i.e., MAXATF(2): manuf(1,2)+manuf(2,2) < MAXPROD(2)
```

An obvious way to make the model more general is to incorporate more than two products and more than two factories. The LET section in XPRESS-MP enables you to introduce parameters so that the model can be altered easily.

```
LET NProd = 2      ! The number of products
LET NFact = 2      ! The number of factories
```

```
VARIABLES          ! Introduce the variables
manuf(NProd,NFact) ! An NProd by Nfact array manuf(p,f)
```

```
TABLES             ! NOTE: there are no data in these
MAXPROD(NFact)     ! tables, so this model will not run
MAXSELL(NProd)
PC(NProd,NFact)
```

```
CONSTRAINTS
MAXMK(p=1:NProd): SUM(f=1:NFact)manuf(p,f) < MAXSELL(p)
MAXATF(f=1:NFact): SUM(p=1:NProd)manuf(p,f) < MAXPROD(f)
PROFITC: SUM(p=1:NProd,f=1:NFact)PC(p,f)*manuf(p,f) $
```

```
GENERATE
```

---

To change the number of products or factories we need only change the first two lines.

We have cheated in the above formulation as we skipped over how to get the correct data into the tables we defined. Tables are initially filled with zero entries until we put data into them, which we normally do with a DISKDATA section (we can also use a DATA section - see *3.1 Example 1: A Simple Production Planning Exercise*, or *DATA Section* in chapter 4, *The mp-model Model Builder*, of the *XPRESS-MP Reference Manual*).

If the data for each table have been assembled in separate ASCII files then the lines:

```
DISKDATA
  MAXPROD = maxprod.dat
  MAXSELL = maxsell.dat
  PC      = pc.dat
```

will fill the tables with the data in the files specified. For instance, the MAXPROD table would be filled with the contents of the file maxprod.dat.

The files must contain comma separated values in ASCII. For instance, pc.dat might contain:

```
!  F1    F2
    50.0,  47.0    ! P1
    125.0, 132.0    ! P2
```

Note that data files too can have comments if preceded by an "!".

Data may be entered into XPRESS-MP from a wide variety of sources, for example, spreadsheets and databases. An example of how to do this is given in *Chapter 5*. For more details refer to chapter 5, *Extending mp-model Using XPRESS-Connect*, in the *XPRESS-MP Reference Manual*.

Even though we have now specified that the PC table data should be read in from a file the situation is not satisfactory, as the PC data depend in turn upon the data in the tables SP, RMP, RMREQ and VC. If these change it would be a nuisance to have to recalculate the PC table or the data in pc.dat by hand. We can solve this problem by using the primary data, and calculating the secondary data PC in the model. Let us insert the extra lines:

---

```

LET NRMat = 2    ! Number of raw materials
TABLES
  SP(NProd)
  RMP(NRMat)
  RMREQ(NProd,NMat)
  VC(NProd,NFact)

DISKDATA
  SP      = sp.dat
  RMP     = rmp.dat
  RMREQ   = rmreq.dat
  VC      = vc.dat

```

to define and fill tables SP, RMP, RMREQ and VC.

The equation that defined the contents of table PC was:

$$PC_{p,f} = SP_p - \left( \sum_{r=1}^{NRMat} RMREQ_{p,r} RMP_r \right) - VC_{p,f}$$

which becomes, in XPress-MP notation:

```

ASSIGN
PC(p=1:NProd,f=1:NFact) = &
  SP(p) - SUM(r=1:NRMat) RMREQ(p,r) * RMP(r) - VC(p,f)

```

Note that the continuation character & has been used to split a long line.

The ASSIGN statement says "for all values of p from 1 to NProd and for f from 1 to NFact, set PC(p,f) to the expression given". The summation operator (SUM) can be used in an expression.

Putting together the different parts, we can assemble a complete model:

```

LET NProd = 2    ! The number of products
LET NFact = 2    ! The number of factories
LET NRMat = 2    ! Number of raw materials

TABLES
  MAXPROD(NFact)
  MAXSELL(NProd)
  PC(NProd,NFact)
  SP(NProd)
  RMP(NRMat)
  RMREQ(NProd,NMat)
  VC(NProd,NFact)

DISKDATA
  MAXPROD = maxprod.dat
  MAXSELL = maxsell.dat

```

---

```

SP      =  sp.dat
RMP     =  rmp.dat
RMREQ  =  rmreq.dat
VC      =  vc.dat

ASSIGN
PC(p=1:NProd,f=1:NFact) =  &
    SP(p) - SUM(r=1:NRMat)RMREQ(p,r)*RMP(r) - VC(p,f)

VARIABLES
    manuf(NProd,NFact) ! An NProd*Nfact array manuf(p,f)

CONSTRAINTS
    MAXMK(p=1:NProd): SUM(f=1:NFact)manuf(p,f) < MAXSELL(p)
    MAXATF(f=1:NFact): SUM(p=1:NProd)manuf(p,f) < MAXPROD(f)
    PROFITC: SUM(p=1:NProd,f=1:NFact)PC(p,f)*manuf(p,f) $

GENERATE

```

We run XPRESS-MP just as before.

## 4.5 Phase 3: Multi-Period Modelling

The purpose of this section is to demonstrate how the simple model we have constructed may be enhanced to cover Coco's operations over several quarters. Coco now wants to plan for the next four quarters and has established the following data tables, which depend on the quarter.

Selling prices of product by quarter:

	Q <sub>1</sub>	Q <sub>2</sub>	Q <sub>3</sub>	Q <sub>4</sub>	
P <sub>1</sub>	400	380	405	350	Table <i>SPT</i>
P <sub>2</sub>	410	397	412	397	

Raw material prices by quarter:

	Q <sub>1</sub>	Q <sub>2</sub>	Q <sub>3</sub>	Q <sub>4</sub>	
R <sub>1</sub>	100	98	97	100	Table <i>RMPT</i>
R <sub>2</sub>	200	195	198	200	

Maximum sales levels for the two products by quarter:

	Q <sub>1</sub>	Q <sub>2</sub>	Q <sub>3</sub>	Q <sub>4</sub>	
P <sub>1</sub>	650	600	500	400	Table <i>MAXSELLT</i>
P <sub>2</sub>	600	500	300	250	

The following tables do not depend on time but are repeated here for completeness.

---

Raw material requirement of products:

	$R_1$	$R_2$	
$P_1$	1.0	0.5	Table <i>RMREQ</i>
$P_2$	1.3	0.4	

Variable costs:

	$F_1$	$F_2$	
$P_1$	150	153	Table <i>VC</i>
$P_2$	75	68	

The maximum total amounts of product that can be made at each factory:

$F_1$	400	Table <i>MAXPROD</i>
$F_2$	500	

It is possible to store raw material and final product from quarter to quarter at a cost of \$2 per tonne per quarter for final product and \$1 per tonne per quarter for raw material. At each factory there is a limited storage capacity for raw material between quarters of 300 tonnes in total. Any amount of final product can be stored.

Currently Coco has stocks of raw materials (in tonnes) of:

	$F_1$	$F_2$	
$R_1$	100	150	Table <i>INIM</i>
$R_2$	50	100	

and of finished product (in tonnes) of:

	$F_1$	$F_2$	
$P_1$	50	100	Table <i>INIP</i>
$P_2$	50	50	

Coco has no constraints on stock levels at the end of the last quarter.

We need to determine:

- how much of each product we manufacture at each factory in each time period;
- how much we should store;
- how much to sell; and
- how much of the two raw materials to buy each quarter.

The usage of raw material is conditional upon how much product we manufacture, but we can store material for use in later quarters.

So that we can generalise the model later to refer to any number of time periods, define a parameter  $NT$  equal to the number of time periods (quarters):

LET  $NT=4$

---

Decision variables are:

$sp(NProd, NFact, NT+1)$ ,	the stock level of final product $p$ in factory $f$ at the start of period $t$ ;
$sm(NRMat, NFact, NT+1)$ ,	the stock level of raw material $r$ in factory $f$ at the start of period $t$ ;
$sell(NProd, NFact, NT)$ ,	the amount of product $p$ sold from factory $f$ in period $t$ ;
$manuf(NProd, NFact, NT)$ ,	the amount of product $p$ made in factory $f$ in period $t$ ; and
$buy(NRMat, NFact, NT)$ ,	the quantity of raw material $r$ bought for factory $f$ in period $t$ .

Some of these variables have a last dimension (the time dimension) of  $NT+1$  rather than  $NT$  as might be expected. This is because it is convenient to consider an  $(NT+1)^{th}$  time period as being the end of the  $NT^{th}$  time period, so expressions involving final stock then refer to the  $(NT+1)^{th}$  period.

The definition of the tables is now:

```

TABLES
  RMREQ(NProd,NRMat)  ! Requirement of product p for
                      !   raw material r
  MAXSELLT(NProd,NT)  ! Maximum amount of p that can
                      !   be sold in period t
  MAXPROD(NFact)      ! Maximum amount factory f can
                      !   make
  SPT(NProd,NT)        ! Selling price of product p in
                      !   period t
  RMPT(NRMat,NT)       ! Price of raw material r in
                      !   period t
  VC(NProd,NFact)      ! Variable cost of product p in
                      !   factory f
  INIM(NRMat,NFact)    ! Initial raw material r stock
                      !   level in factory f
  INIP(NProd,NFact)    ! Initial final product p stock
                      !   level in factory f

```

and of the variables:

```

VARIABLES
  sp(NProd,NFact,NT+1) ! Stock level of final product
                      !   p in fact. f at start period t
  sm(NRMat,NFact,NT+1) ! Stock level of raw material r
                      !   in fact f at start of period t
  sell(NProd,NFact,NT) ! Amount of product p sold from
                      !   factory f in period t
  manuf(NProd,NFact,NT) ! Amount of product p made in
                      !   factory f in period t

```

---

buy(NRMat,NFact,NT) ! Quantity of raw material r  
! bought for fact f in period t

The mathematical formulation of the objective function and constraints follows.

Maximise

$$\begin{aligned}
& \sum_{p=1}^{NProd} \sum_{f=1}^{NFact} \sum_{t=1}^{NT} SPT_{p,t} sell_{p,f,t} \\
& - \sum_{p=1}^{NProd} \sum_{f=1}^{NFact} \sum_{t=1}^{NT} VC_{p,f} manif_{p,f,t} \\
& - \sum_{r=1}^{NRMat} \sum_{f=1}^{NFact} \sum_{t=1}^{NT} RMPT_{r,t} buy_{r,f,t} \\
& - \sum_{p=1}^{NProd} \sum_{f=1}^{NFact} \sum_{t=2}^{NT+1} 2sp_{p,f,t} \\
& - \sum_{r=1}^{NRMat} \sum_{f=1}^{NFact} \sum_{t=2}^{NT+1} 1sm_{r,f,t}
\end{aligned}$$

subject to

- (A)  $sp_{p,f,t+1} = sp_{p,f,t} - sell_{p,f,t} + manif_{p,f,t}, \forall p, f, t$
- (B)  $sm_{r,f,t+1} = sm_{r,f,t} + buy_{r,f,t} - \sum_{p=1}^{NProd} RMREQ_{p,r} manif_{p,f,t}, \forall r, f, t$
- (C)  $\sum_{f=1}^{NFact} sell_{p,f,t} \leq MAXSELLT_{p,t}, \forall p, t$
- (D)  $\sum_{p=1}^{NProd} manif_{p,f,t} \leq MAXPROD_f, \forall f, t$
- (E)  $\sum_{r=1}^{NRMat} sm_{r,f,t} \leq 300, \forall f, t = 2, \dots, NT+1$
- (F)  $sp_{p,f,1} = INIP_{p,f}, \forall p, f$
- (G)  $sm_{r,f,1} = INIM_{r,f}, \forall r, f$
- (H) All variables non - negative

The symbol " $\forall$ " is to be read as "for all". So, for example,  $\forall p, f$  means "for  $p$  from 1 to  $NProd$  and for  $f$  from 1 to  $NFact$ ".  $\forall t$  means "for all  $t$  from 1 to  $NT$ ". If  $t$  is to range from 2 to  $NT+1$  then this is explicitly shown.

---

The objective function gives the profit as follows:

revenue from sales (unit price  $\times$  amount sold)  
minus variable manufacturing costs (unit cost  $\times$  amount made)  
minus raw material costs (unit cost  $\times$  amount bought)  
minus storage cost of products (unit cost  $\times$  amount stored)  
minus storage cost of raw materials (unit cost  $\times$  amount stored)

In words the constraints say:

- (A) the stock level at the beginning of period  $t+1$  is the stock level at the start of period  $t$  less what we sell plus what we make.
- (B) the raw material stock level at the beginning of period  $t+1$  is the stock level at the start of period  $t$ , plus what we buy, less what we use in making products.
- (C) we cannot sell in total more than  $MAXSELLT$ .
- (D) we cannot manufacture more in total than we have manufacturing capacity for.
- (E) we cannot store more than 300 tonnes of raw material at the end of any period. (Equivalently, we cannot have more than 300 tonnes of raw material in stock at the start of periods 2, ...,  $NT+1$ ).
- (F) the initial product stock levels are as specified.
- (G) the initial raw material stock levels are as specified.
- (H) we cannot produce negative quantities of a product.

In XPRESS-MP notation the constraints are written as:

```
CONSTRAINTS
! Objective function (maximise)
PROFITC    &
SUM(p=1:NProd,f=1:NFact,t=1:NT) SPT(p,t)*sell(p,f,t) &
-SUM(p=1:NProd,f=1:NFact,t=1:NT) VC(p,f)*manuf(p,f,t) &
-SUM(r=1:NRMat,f=1:NFact,t=1:NT) RMPT(r,t)*buy(r,f,t) &
-SUM(p=1:NProd,f=1:NFact,t=2:NT+1) 2*sp(p,f,t)          &
-SUM(r=1:NRMat,f=1:NFact,t=2:NT+1) 1*sm(r,f,t)          $

! Final product stock balance
SB(p=1:NProd,f=1:NFact,t=1:NT):                          &
    sp(p,f,t+1) = sp(p,f,t)-sell(p,f,t)+manuf(p,f,t)

! Raw material stock balance
SR(r=1:NRMat,f=1:NFact,t=1:NT):                          &
    sm(r,f,t+1) = sm(r,f,t) + buy(r,f,t)                  &
    -SUM(p=1:NProd)RMREQ(p,r)*manuf(p,f,t)

! Selling limit
MXS(p=1:NProd,t=1:NT):                                    &
```

---

```

SUM(f=1:NFact) sell(p,f,t) < MAXSELLT(p,t)

! Maximum production limit
MXM(f=1:NFact,t=1:NT):                                     &
SUM(p=1:NProd) manuf(p,f,t) < MAXPROD(f)

! Raw material storage
SCRM(f=1:NFact,t=2:NT+1): SUM(r=1:NRMat)sm(r,f,t) < 300

BOUNDS
! Initial product levels
sp(p=1:NProd,f=1:NFact,1) = INIP(p,f)

! Initial raw material levels
sm(r=1:NRMat,f=1:NFact,1) = INIM(r,f)

GENERATE

```

An unfamiliar feature is the BOUNDS section, which is used to set simple upper and lower bounds, or fixed values, for a variable or set of variables. The format is superficially similar to the ASSIGN statement. If we consider the last row, the entry says "for all raw materials  $r$ , and for all factories  $f$ , set  $sm(r, f, 1)$ , i.e., the initial raw material stock level, equal to  $INIM(r, f)$ ".

If a variable has an upper bound  $U$  then this is indicated by using the notation:

$$x < U$$

if it has a lower bound  $L$ , by specifying:

$$x > L$$

or if it is to be fixed to  $E$  by specifying:

$$x = E$$

These can be thought of as constraints on single variables where we do not have to give a constraint name. However, limits can be processed by the optimiser more efficiently as bounds than as constraints in the CONSTRAINTS section.

Some constants are still hard-wired into the model. Strictly speaking they should be removed but since they will not influence the solution, we can leave their removal to another time.

Putting all the parts of the model together we get:

```

LET NT      = 4
LET NProd   = 2
LET NRMat   = 2
LET NFact   = 2

TABLES

```

---

```

RMREQ(NProd,NRMat)    ! Requirement of product p for
                      !   raw material r
MAXSELLT(NProd,NT)    ! Maximum amount of p that can
                      !   be sold in period t
MAXPROD(NFact)        ! Maximum amount factory f can
                      !   make
SPT(NProd,NT)         ! Selling price of product p in
                      !   period t
RMPT(NRMat,NT)        ! Unit price of raw material r
                      !   in period t
VC(NProd,NFact)       ! Variable cost of product p in
                      !   factory f
INIM(NRMat,NFact)     ! Initial raw material r stock
                      !   level in factory f
INIP(NProd,NFact)     ! Initial final product p stock
                      !   level in factory f

DISKDATA              ! Get data into tables
RMREQ = rmreq.dat
MAXSELLT = maxsell.dat
MAXPROD = maxprod.dat
SPT = spt.dat
RMPT = rmpt.dat
VC = vc.dat
INIM = inim.dat
INIP = inip.dat

VARIABLES
sp(NProd,NFact,NT+1) ! Stock level of final product p
                      !   in factory f at start of period t
sm(NRMat,NFact,NT+1) ! Stock level of raw material r
                      !   in factory f at start period t
sell(NProd,NFact,NT) ! Amount of product p sold from
                      !   factory f in period t
manuf(NProd,NFact,NT)! Amount of product p made
                      !   in factory f in period t
buy(NRMat,NFact,NT)  ! Quantity of raw material r
                      !   bought for factory f in period t

CONSTRAINTS
! Objective function (maximise)
PROFITC:                                                     &
SUM(p=1:NProd,f=1:NFact,t=1:NT) SPT(p,t)*sell(p,f,t) &
-SUM(p=1:NProd,f=1:NFact,t=1:NT) VC(p,f)*manuf(p,f,t) &
-SUM(r=1:NRMat,f=1:NFact,t=1:NT) RMPT(r,t)*buy(r,f,t) &
-SUM(p=1:NProd,f=1:NFact,t=2:NT+1) 2*sp(p,f,t)             &
-SUM(r=1:NRMat,f=1:NFact,t=2:NT+1) 1*sm(r,f,t)             $

```

---

```

! Final product stock balance
SB(p=1:NProd,f=1:NFact,t=1:NT):
    sp(p,f,t+1) = sp(p,f,t)-sell(p,f,t)+manuf(p,f,t) &

! Raw material stock balance
SR(r=1:NRMat,f=1:NFact,t=1:NT):
    sm(r,f,t+1) = sm(r,f,t) + buy(r,f,t) &
    -SUM(p=1:NProd)RMREQ(p,r)*manuf(p,f,t) &

! Selling limit
MXS(p=1:NProd,t=1:NT):
    SUM(f=1:NFact) sell(p,f,t) < MAXSELLT(p,t) &

! Maximum production limit
MXM(f=1:NFact,t=1:NT):
    SUM(p=1:NProd)manuf(p,f,t) < MAXPROD(f) &

! Raw material storage
SCRM(f=1:NFact,t=2:NT+1): SUM(r=1:NRMat)sm(r,f,t) < 300

BOUNDS
! Initial product levels
sp(p=1:NProd,f=1:NFact,1) = INIP(p,f)

! Initial raw material levels
sm(r=1:NRMat,f=1:NFact,1) = INIM(r,f)

GENERATE

```

## 4.6 Phase 4: Introducing Fixed Costs

Coco realises that it has forgotten to take into account the fixed costs per quarter associated with a factory being open, independent of the production level. The costs are \$50,000 for factory  $F_1$  and \$63,000 for factory  $F_2$ . Apart from affecting the overall profitability of the operation, these costs do not change the policy that Coco has pursued from the previous models.

We have a table of the cost of the factories being open:

$F_1$	50,000	Table <i>COPEN</i>
$F_2$	63,000	

Putting this data into a file called `copen.dat` we then add the following lines to the relevant sections of the model:

```

TABLES
    COPEN(NFact)

```

---

```
DISKDATA
  COPEN = copen.dat
```

But Coco can now consider closing one or both of the factories for one or more quarters. If a factory is closed for a quarter then a fixed cost of only \$20,000 is incurred that quarter. If the factory is closed it cannot produce in that quarter.

We define some variables  $open(NFact, NT)$ , which are 1 if factory  $f$  is open in period  $t$ , else 0. These are *binary (0/1) variables* and are declared as such in the BOUNDS section.

```
VARIABLES
  open(NFact, NT)
  .....
BOUNDS
  open(f=1:NFact, t=1:NT) .BV. ! The open variables
                                ! are binaries
```

The .BV. indicates that the open variables are binary variables. Incidentally, we can declare *integer variables* in a similar way:

```
x(i=1:I) .UI. 6 ! The x variables
               ! are general integers
```

This will declare the  $x$  variables as being integers taking one of the values 0,1,...,6. However, this is not required in Coco's model.

Allowing for the fact that we incur a cost of \$20000 whether the factory is open or not we get an increased cost of  $(COPEN(f)-20000)$  if  $open(f,t)$  is 1. As long as we remember that we incur an additional fixed cost of \$20000 per factory per quarter (i.e.,  $20000*NFact*NT$ ), we can model the increased fixed cost as an extra term in the objective function:

```
-SUM(f=1:NFact, t=1:NT) (COPEN(f)-20000)*open(f, t)
```

and we have to modify the maximum production limit constraints to:

```
MXM(f=1:NFact, t=1:NT) :
  SUM(p=1:NProd)manuf(p, f, t) < MAXPROD(f)*open(f, t) &
```

If  $open(f,t)$  is zero (i.e., the factory is closed) then all the *manuf* variables for that  $f$  and  $t$  must be zero since each one is non-negative and their sum is 0. If  $open(f,t)$  is 1 (i.e., the factory is open), then the total production is limited by  $MAXPROD(f)$ , as before.



Running XPRESS-MP takes the now familiar form:

```
mp-model cocol
INPUT
```

but running mp-opt is slightly different as we now have a Mixed Integer Programming (MIP) problem:

```

mp-opt cocol
INPUT
MAXIMISE
GLOBAL
FPRINT
QUIT

```

As before, the MAXIMISE command finds the linear programming optimal solution to the problem, ignoring the fact that the open variables must be either 0 or 1. The extra command GLOBAL requests mp-opt to conduct a search for the MIP optimal solution. The search uses the Branch and Bound technique, which is described in detail in the *XPRESS-MP Reference Manual*.

When the search finishes, mp-opt will have found the best (most profitable) way of setting both the binary and the normal continuous variables. The optimal solution is available to the FPRINT (or PRINT) commands.



Since we now have an Integer Programming problem, when we use **Visual XPRESS** we first click Run Solve LP to solve the underlying linear program, and then click Run Solve Global to solve the MIP. When the search finishes, **Visual XPRESS** will have found the optimal solution setting both the binary the continuous variables. As usual, Run Solution Print File will display the solution.

## 4.7 Phase 5: Permanent Closure?

Coco now realises that it could close down a factory permanently. If it does so it does not incur the fixed cost of \$20000 mentioned previously but it can never re-open the factory. The objective function is now:

```

! Objective function (maximise)
PROFITC:
SUM(p=1:NProd,f=1:NFact,t=1:NT) SPT(p,t)*sell(p,f,t) &
-SUM(p=1:NProd,f=1:NFact,t=1:NT) VC(p,f)*manuf(p,f,t) &
-SUM(r=1:NRMAT,f=1:NFact,t=1:NT) RMPT(r,t)*buy(r,f,t) &
-SUM(p=1:NProd,f=1:NFact,t=2:NT+1) 2*sp(p,f,t) &
-SUM(r=1:NRMAT,f=1:NFact,t=2:NT+1) 1*sm(r,f,t) &
-SUM(f=1:NFact,t=1:NT) COPEN(f)*open(f,t) $

```

since the cost COPEN is only incurred if the factory is open. However once the factory is closed (not open) it stays that way forever (we don't consider the possibility of reopening a factory at all). The following *logical constraints* perform this:

```

! If factory closed once, always closed
O(f=1:NFact,t=1:NT-1): open(f,t+1) < open(f,t)

```

To see why this is, suppose that factory 1 closes in period 2, i.e., that *open(1,2)* is 0.

---

The constraints read:

```
O(1,1):  open(1,2) < open(1,1)
O(1,2):  open(1,3) < open(1,2)
O(1,3):  open(1,4) < open(1,3)
```

The  $O(1,2)$  constraint ensures that  $open(1,3)$  is zero if  $open(1,2)$  is zero; and the  $O(1,3)$  constraint similarly ensures that  $open(1,4)$  is zero.

## 4.8 Where Have We Got To?

We have built a complex model in just a few steps. It is multi-product, multi-material and multi-period. It also has a mixed integer programming component to model logical constraints.

We have achieved the desirable objective of separating the model's structure from the data that make up a particular instance of the model. If one of the raw material costs changes, we do not have to change the model, with all the accompanying risks of the user accidentally harming the model structure.

If the data provider decides that he or she wants to maintain the data in another format (for instance, a spreadsheet) then other features of XPRESS-MP will allow access to the data from the new source.

What is more, the ASSIGN section lets you calculate some of the data required by the model from the raw data. You do not have to depend upon the data provider remembering to perform some calculations to update dependent data. This is exemplified by the calculation of the *PC* table, where the profit contribution (*PC*) depended on the raw data *SP* (selling price), *RMREQ* (raw material required), *RMP* (raw material prices), and *VC* (the variable production costs). In practice it is likely that each of these four data files will be the responsibility of a different department in the company, but no-one will be responsible for calculating the *PC* table. So, managerially, the correct place to bring them together is inside the model.



---

## 5 More Advanced Modelling Features

### 5.1 Overview

This chapter introduces some more advanced features of the modelling language in `mp-model` and **Visual XPRESS**. We shall not attempt to cover all its features or give the detailed specification of their formats. These are covered in greater depth in the *XPRESS-MP Reference Manual*.

The main areas not yet covered in any detail are

- conditional generation
- importing data
- sparse data
- displaying data
- non-numeric indices
- built in functions
- basic reporting of results

The following sections deal with each of these in turn.

### 5.2 Conditional Generation of Variables and Constraints

Suppose we wish to apply an upper bound to some but not all members of a set of variables  $x_i$ . There are *MAXI* members of the set. The upper bound to be applied to  $x_i$  is  $U_i$ , but it is only to be applied if the entry in the data table  $TAB_i$  is greater than 20. This can be expressed thus:

```
BOUNDS          ! Dependent on TAB(i)
x(i=1:MAXI | TAB(i) > 20) < U(i)
```

If the bound did not depend on the value in  $TAB_i$  then the line would have read:

```
BOUNDS
x(i=1:MAXI) < U(i)
```

The vertical bar (|) character, followed by a *logical expression*, is to be read as "to be done if the following logical expression is true". So the line in the first example is to be

---

read: "for the x variables from i=1 to i=MAXI, if the value of TAB(i) is greater than 20 then there is an upper bound of U(i) on the x(i)".

The logical expression can be built up in a way similar to that in high level programming languages. Here are some examples.

```
i .eq. 10
i == 10
i .le. 10 .and. TAB(i) .ne. 0
i <= 10 .and. TAB(i) <> 0
i+2*j > t+TAB(i)
```

The first and second logical expressions above are only true if i is equal to 10. So:

```
CONSTRAINTS
CC(i=1:MAXI | i == 10) SUM(j=1:i) x(j) > i
```

has the same effect as:

```
CONSTRAINTS
CC(i=10:10): SUM(j=1:10) x(j) > i
```

See **4.3 The Building Blocks of mp-model** in the *XPRESS-MP Reference Manual* for details of the formal rules.

Logical expressions do not have to involve dummy subscripts. They can, for instance, involve parameters or raw data items:

```
LET IFDO = 1
CONSTRAINTS
UP(k=1:KK | IFDO <> 0): SUM(j=1:10) x(j,k) < TT(k)
S( | IFDO == 0 ): SUM(j=1:10,k=1:KK) x(j,k)=1
```

If the parameter IFDO is non-zero then the UP constraints are generated. If it is zero then the single constraint S is generated. Notice especially the format of the last constraint, where we must add some parentheses to a non-subscripted constraint to be able to incorporate the | construct.

As we have seen, the conditional generation facility can be applied in the BOUNDS and CONSTRAINTS sections, but it can also be applied in the VARIABLES and SETS sections.

In particular, the existence of variables can be made conditional. For instance:

```
LET NN = 10
TABLES
TAB(NN)
DATA
TAB(1) = 1, 2, 3, 0, -1, 6, -7, 8, -9, 10
VARIABLES
x(i=1:NN | TAB(i) == i )
```

---

will define just the variables  $x(1)$ ,  $x(2)$ ,  $x(3)$ ,  $x(6)$ ,  $x(8)$  and  $x(10)$ .

This feature can be particularly useful, because any condition included in the definition of the variables does not need to be repeated whenever the variables are used. Variables  $x(i)$  can be referred to in subsequent sections for all  $i$  between 1 and NN. If  $x(i)$  has not been defined (e.g.,  $i=4$ ) then any reference to it is ignored.

For example:

```
BOUNDS
    x(i=1:NN) > 50
```

Creates bounds only on the defined variables  $x(1)$ ,  $x(2)$ ,  $x(3)$ ,  $x(6)$ ,  $x(8)$  and  $x(10)$  just as if we had entered the bounds  $x(i=1:NN \mid \text{TAB}(i)=i)$ . Putting the condition in the variable definition statement removes a lot of clutter from the constraints and bounds and makes the model easier to read.

## 5.3 Importing Data

There are obvious benefits to be gained from separating the form, or structure, of a model from the particular data that give a model instance. The XPRESS-MP modelling language encourages this modelling principle and incorporates a powerful set of facilities for importing data.

It is possible to enter data into the modeller's data tables in three main ways:

- by use of the DATA section
- by use of the DISKDATA section
- through the ASSIGN section.

### 5.3.1 The DATA Section

The first of these, the DATA section, is a somewhat old-fashioned method whereby the data are embedded in the model. Since it does not try to separate the structure of a model from the data its use is limited to the creation of small test examples or the development of prototype models. See **3.1 Example 1: A Simple Production Planning Exercise**.

To insert data into a one dimensional table we might do this:

```
TABLES
    TAB(5)
DATA
    TAB(1) = 6, -2, 0, 3
```

The format of the DATA statement is as follows: on the left hand side of the equal sign we put the first position in the table where data is to be read; and on the right hand side of the equal sign we place the data values which are to be read into this position and

---

subsequent positions in the table. So in the example above `TAB(1)` receives the value 6, `TAB(2)` receives the value -2, `TAB(3)` gets 0, and `TAB(4)` receives the value 3. `TAB(5)` retains the zero value with which it was initialised when it was declared in the `TABLES` section.

Note that the first element in a table is element number 1 and *not* element number 0.

In a two dimensional table we might have something like the following:

```
TABLES
  PP(2,2)
DATA
  PP(1,1) = 1, 2 ! line 4
  PP(2,1) = 3, 4 ! line 5
```

The fourth line in the above example fills up the first row of the table `PP` and the fifth line fills up the second row of the table. The easy way to remember this is that the block of numbers on the right hand side of the equality signs would form the entries if you considered table `PP` to be a matrix.

One point to note is that the data items do not have to be constants, but can in fact be any expression. For instance, in the example above we might have:

```
LET FRED=66
TABLES
  PP(2,2)
DATA
  PP(1,1) = 1, 2*3.1
  PP(2,1) = 3, 4*FRED
```

The expression in the first row, second column is evaluated as the data are read into the table and so if we inspected the contents of `PP(1,2)` we would see the value 6.2. The expression in the second row and second column is evaluated as 264 and stored in `PP(2,2)`. Parameterising a model in this way is sometimes useful. Of course, the data cannot depend on decision variables.

It is possible to use the `DATA` statement to input data into three and more dimensional tables, but it gets progressively more tricky to do so as the number of dimensions increases. For full details see *DATA Section* in chapter 4, *The mp-model Model Builder*, of the *XPRESS-MP Reference Manual*.

### 5.3.2 The DISKDATA Section

By far the most common way to get data into a model is by using the `DISKDATA` section, which enables data to be imported from external sources. These sources are generally text files, databases or spreadsheets. Text data (i.e., ASCII files) are catered for by routines built into the `XPRESS` modellers as we have already seen in **3.2 Example 2: A Blending Example**. Other sources of data can be accessed by means of the `XPRESS Connect` technology.

---

Let us consider text files first. These must be in the host operating system's character set, usually ASCII, but EBCDIC for some mainframes. These must not contain control characters, so if you use a word processor to create the file you must make sure you save it in text mode.

Suppose that we have a file `bob.dat` into which we have placed the data:

```
1.6, 3.4
2.8, -7.6
```

and that we wish to read this into the 2 by 2 table `JOE`. We can do this by statements of the following form:

```
TABLES
JOE(2,2)
DISKDATA
JOE = bob.dat
```

The format of the `DISKDATA` statement is easy to understand: on the left of the equal sign is the name of the table *into which* the data are to be read; and on the right hand side is the name of the file *from which* the data are to be read. The number of rows and columns in the data should match the number of rows and columns in the table.

A one dimensional table can be filled in a similar manner:

```
TABLES
JANE(6)
DISKDATA
JANE = c:\temp\zoe.in
```

The file `zoe.in` contains the line:

```
6.1, 0, 0, 1.0, 1, 1
```

which will fill all the entries in table `JANE`. Note that you can specify paths as part of the filename.

Again, it is possible for some or all of the data items in the file to be replaced by expressions, which are evaluated as the data is read in. Three and higher dimensional tables are catered for. See the ***XPRESS-MP Reference Manual*** for details.

It is quite easy for a sophisticated computer user to create and maintain data tables in text files but we have found that a much better data storage medium is provided by spreadsheets, so there is a facility in `XPRESS-MP` (called `XPRESS-Connect ODBC`), whereby the contents of ranges within spreadsheets may be read into data tables. It requires an additional authorisation in your `XPRESS-MP` licence.

Please note that if you are going to work through the examples in the following section, you will have to have access to Excel.

---

Let us suppose that in a spreadsheet called `myss.xls` you have inserted the following into the cells indicated:

	A	B	C
1			
2		First	Second
3		6.2	1.3
4		-1.0	16.2
5		2.0	-17.9

and called the range B2:C5 `MyRange`.

We will use ODBC to extract these data into a table `TOM(3,2)`. The ***Reference Manual*** gives detailed instructions on how to use ODBC, so we give just a brief summary here.

- In Windows' Control Panel, select 32-bit ODBC and set up a User Data Source called `MyExcel`, by clicking Add, selecting Microsoft Excel Driver (\*.xls), and filling in the ODBC Microsoft Excel Setup dialog. Click Options >> and clear the Read Only check box.

The following statements will set up the `TOM` table in `XPRESS-MP` and fill it with the data from the Excel range `MyRange`.

```
TABLES
  TOM(3,2)
CONNECT ODBC, 'DSN=MyExcel;DBQ=myss.xls'
DISKDATA -c
  TOM = 'select * from MyRange'
DISCONNECT
```

The `CONNECT` statement requires an ODBC DSN (which we just set up) and an Excel spreadsheet. Data can then be read in by the `DISKDATA -c` command, where the `XPRESS-MP` table to be filled is on the left of the `=` sign, and the data to be extracted on the right. A spreadsheet range must have a top row with column titles (here we used `First` and `Second` as the titles).

The ODBC statement `'select * from MyRange'` says "select everything from the range called `MyRange`". ODBC uses SQL, and it is possible to have much more complex selection statements than the ones we have used.

See chapter 5, *Extending mp-model Using XPRESS-Connect*, in the ***XPRESS-MP Reference Manual*** for more details on ODBC.

### 5.3.3 Helpful Tip: Sizing TABLES for Spreadsheet Data

There is a trick that we have found to be very useful in practice. The `XPRESS-MP` modeller is interpreted, i.e., it does the evaluation of what it sees at the point that it sees

it. This is particularly helpful when you want to make dynamic adjustments to the sizes of tables. Below we develop a set of commands that we use repeatedly when we are feeding XPRESS-MP with data from a spreadsheet that obtain the sizes of tables directly from the data source.

Suppose that we have set up some data in a spreadsheet `ssxmpl` to represent the resource usage of some raw materials by some products. We want to be able to allow for the number of raw materials and the number of products changing. Diagrammatically, we have decided to lay out this part of the spreadsheet as follows:

	RawMat1	RawMat2	RawMat3	RawMat4
Product1				
Product2				
Product3				
...				

and to call the range, including the row containing raw material names (but not the column containing product names), `USAGE`. If an extra product is introduced, then `USAGE` gets bigger by one row; if an extra raw material is used then the number of columns increases by one. These changes have to be mirrored in the XPRESS-MP formulation.

We construct a small region of the spreadsheet and name it `SIZES`. Into it we put the numbers that characterise the problem - in this case the number of products and the number of raw materials - the parameters. It is important that these numbers are not "hard-wired" in, but that we let the spreadsheet calculate them for itself. In Excel it would be:

Number of Products	Number of Raw Materials
<code>=ROWS(USAGE)-1</code>	<code>=COLUMNS(USAGE)</code>

The reduction by 1 allows for the row which just contains the raw material names.

Now the commands below might form the introductory part of a model.

---

```

! Excel example                                ! Line 1
CONNECT ODBC, 'DSN=MyExcel;DBQ=ssxmpl.xls'      ! Line 2
TABLES                                           ! Line 3
  MYSIZES(2)                                     ! Line 4
DISKDATA -c                                     ! Line 5
  MYSIZES = 'select * from SIZES'               ! Line 6

LET Nprod = MYSIZES(1)                          ! Line 8
LET Nrm = MYSIZES(2)                           ! Line 9

TABLES                                           ! Line 11
  PneedsR(Nprod,Nrm)                          ! Line 12

DISKDATA -c                                     ! Line 14
  PneedsR = 'select * from USAGE'              ! Line 15

```

Lines 3 and 4 define a little table which is going to hold the model sizing parameters. Lines 5 and 6 read the contents of the spreadsheet range SIZES into our little XPRESS-MP table MYSIZES. Then in lines 8 and 9 we set up some parameters internal to XPRESS-MP which hold the values. This is not a required step - it is just a lot clearer to refer later in the model to Nprod and Nrm rather than MYSIZES(1) and MYSIZES(2). It is likely, for instance, that we will need to define some decision variables dimensioned by Nprod.

Then in lines 11 and 12 we define an Nprod by Nrm table to hold the requirements of the products for the raw materials. This is the key step. If the size of USAGE changes in the spreadsheet then via the range SIZES and our XPRESS-MP table MYSIZES, the dimensions of the PneedsR table will adjust automatically. Then lines 14 and 15 will read the USAGE range into the XPRESS-MP table array PneedsR of the correct dimensions.

### 5.3.4 The ASSIGN Section

The third common way to obtain data in XPRESS-MP is to use the ASSIGN section to calculate the values of data you require from values in other tables.

We saw an example of this in the chapter on the Coco company (*Section 4.3*), where the statements below were used to calculate the profit contributions of making particular products at various factories. The tables SP, RMREQ, RMP and VC held primary data, and the ASSIGN section was used to calculate the values of the dependent data:

```

ASSIGN
  PC(p=1:NProd,f=1:NFact) = &
    SP(p,f) - SUM(r=1:NRMat) RMREQ(p,r) * RMP(r) - VC(p,f)

```

---

On the left of the equal sign is a specification of those table entries that are to be set, whilst on the right is an arithmetic expression. The usual rules of arithmetic apply, and the SUM operator can be used.

## 5.4 Sparse Data Tables

Almost all large scale LP and MIP problems have a property known as *sparsity*, that is each variable appears with a non-zero coefficient in a very small fraction of total set of constraints. Often this property is reflected in the data tables used in the model in that many values of the tables are zero. When this happens, it is more convenient to provide just the non-zero values of the data table rather than listing all the values, the majority of which are zero. This is also the easiest way to input data into data tables with more than two dimensions. An added advantage is that less memory is used by the XPRESS-MP modeller.

A data table is defined to be *sparse* by using the `-e` option in the TABLES section where it is defined, and giving the maximum number of non-zero values that the table may have. For example

```
TABLES
  SID(100,100) -e 50
```

defines the two dimensional data table `SID(100,100)`, which may contain at most 50 non-zero data values. Note that without the `-e` option, XPRESS-MP has to set aside space for  $100*100 = 10,000$  data values which uses considerably more memory. The saving would be even more significant for higher dimensional tables. Obviously this has great benefits in larger problems where memory size is becoming a restraining factor. The penalty is a small loss in efficiency, which should not significantly affect performance.

To read data into a sparse data table from a data file, use *sparse data format*. Each data value is specified on a separate line by giving a list of subscripts specifying a single element of the data table, followed by the data value itself. The subscripts and the data value are separated by commas. Of course, zero data values are not specified.

For example, a data file `sid.dat` to initialise the `SID` table defined above might look like

```
12, 76, 639.75
87, 36, 1015.00
55, 30, 824.25
```

The data file is read in using a DISKDATA section in the normal way

```
DISKDATA
  SID = sid.dat
```

---

This specifies three non-zero values:

```
SID(12,76) = 639.75
SID(87,36) = 1015.00
SID(55,30) = 824.25
```

All other elements of SID have the value zero.

Here is a second example showing how to initialise a five dimensional data table

```
TABLES
  PETE(10,50,20,100,55) -e 1000

DISKDATA
  PETE = pete.dat
```

where `pete.dat` contains

```
2, 47, 18, 88, 12, 2205.87
9, 30, 8, 18, 9, 1780.24
5, 24, 2, 61, 31, 4002.33
3, 11, 20, 34, 43, 7550.03
```

We have defined PETE to contain up to 1000 elements, although we have specified just four in the data file. Note that the elements referred to can be in any order. Notice also how the table would require space for  $10*50*20*100*55 = 55,000,000$  values if it was not defined to be sparse.

It is possible to use sparse data format for ordinary (*dense*) data tables and ordinary (*dense*) data format for sparse tables using the `-s` and `-d` options with DISKDATA. Sparse data format can also be combined with the `-c` option to input data from external data sources using ODBC. See the *DISKDATA Section* in chapter 4, *The mp-model Model Builder*, of the *XPRESS-MP Reference Manual* for details.

## 5.5 Displaying Data

Frequently we wish to display the contents of data tables, most usually when checking that data have been imported correctly, or that assignments have been done as intended. XPRESS-MP provides two methods for showing the contents of parameters or data tables. The first of these is the PRINT section, and the second is an extension of the DISKDATA section. PRINT can normally be used in the stand alone module `mp-model` only. See below for a method of displaying data in **Visual XPRESS**.

---

### 5.5.1 The PRINT Section



Suppose we have the following few lines of a model:

```
TABLES
  TAB(5)
  CUSUM(5)
DATA
  TAB(1)=6, -2, 0, 3, 4
ASSIGN
  CUSUM(i=1:5)=SUM(j=1:i) TAB(j)
```

whose intention is to fill up table CUSUM with the partial cumulative sums of the data in table TAB. We can check that we have performed the calculations correctly by entering the PRINT section and displaying some, or all, of the table's values:

```
PRINT
  CUSUM(1)
  CUSUM(5)
```

The expression on each line in the section is evaluated and displayed. We see that CUSUM(1) has the value 6.000000, whilst CUSUM(5) has a value 11.000000.

It is possible to use expressions that are more complicated than simple table entries. For instance, we might say:

```
PRINT
  10.0+CUSUM(1)+CUSUM(5)
```

and see the value 27.000000.

It is also possible to display all the cells in a table:

```
PRINT
  CUSUM
```

displays the values of all the cells in CUSUM.

The PRINT section has the disadvantage that the display can only be sent to the screen (actually the output is also included in the list file – see **3.1.3 Correcting Errors**). If it is necessary to send the output to a file then we can use an extra feature of the DISKDATA section, which also allows values to be displayed from **Visual XPRESS**.

---

### 5.5.2 Using DISKDATA to Output Data

So far we have used DISKDATA to get data from an external file into a table. Its format was, roughly:

```
DISKDATA
  table = file
```

to fill table from file. If we want to do the reverse operation, send the contents of a table to a file, then the format is:

```
DISKDATA -o
  file = table
```

The letter following the minus sign is o, lower case "oh", for Output. For example:

```
TABLES
  TAB(5,3)                                ! Define table TAB
ASSIGN
  TAB(i=1:5,j=1:3) = i+j                 ! Assign data to TAB
DISKDATA -o
  tabvals.dat = TAB                      ! Output TAB to file
```

defines a 5 by 3 table, assigns some data to it so that the entry in the row i column j is i+j, then outputs the contents of the table to the file tabvals.dat.

The data are output as comma separated text. If we inspected tabvals.dat we would see the following:

```
2.000000, 3.000000, 4.000000
3.000000, 4.000000, 5.000000
4.000000, 5.000000, 6.000000
5.000000, 6.000000, 7.000000
6.000000, 7.000000, 8.000000
```

Sparse data tables can also be output to files, but then sparse data format is used and zero values are not included. For example, to output the data table SID defined in **5.4 Sparse Data Tables**, use

```
DISKDATA -o
  sid.out = SID
```

and then sid.out will contain

```
12,      76,      639.750000
55,      30,      824.250000
87,      36,      1015.000000
```

It is also possible to use the DISKDATA -o and ODBC to export results to an external data source such as a spreadsheet or database. See **5.8 Exporting Results** later in this chapter.

---

## 5.6 Index Sets

(The use of Index Sets is relatively advanced, so this section only skims the surface. See *INDICES Section* in chapter 4, *The mp-model Model Builder*, of the *XPRESS-MP Reference Manual* for full details. In particular, care has to be taken that the intermediate matrix file is not ambiguous).

Index sets are very powerful, enabling you to use things other than just the numbers 1, 2, ... , to index arrays of variables and tables. They also contribute to using sparse data tables, that is data whose positions in a table are specified individually by the location in the table and the data value living at that location. Here are some examples of the use of index sets.

### 5.6.1 Example 1: Using index set with data tables and variables

Suppose we have a set ID containing ND depots, and a set IC containing NC customers. In file `cost.dat` we have data as follows:

```
!Depot Customer Cost
BHouse, Cosytec, 23.22
LSpa , Cosytec, 21.22
BHouse, DOI , 16.2
LSpa , DOI , 25.1
BHouse, MStone , 89.1
LSpa , MStone , 92.1
```

giving the per unit cost of sending material from depot  $d$  to customer  $c$ .

```
LET
  ND=2                ! Number of depots
  NC=3                ! Number of customers

INDICES                ! Hold the names of the
  ID(ND)              ! depots and
  IC(NC)              ! customers

TABLES
  COST(ID,IC)         ! unit cost of sending from id to ic

SET DYNINDEX           ! Set up the index names "on the fly"
                        ! See below for explanation.
DISKDATA -s           ! Read data in sparse format
  COST = cost.dat
```

The `SET DYNINDEX` statement tells the modeller to build up the sets of depots (in `ID`) and customers (in `IC`) dynamically - every time a new set member is seen when inputting data it is added to the set. If after the `DISKDATA` statement we issue the commands in `mp-model`

---

```

PRINT
  ID
  IC
  COST

```

we see

```

  ID
  "BHouse  "
  "LSpa    "
  IC
  "Cosytec "
  "DOI     "
  "MStone  "
  COST
    23.220000,    16.200000,    89.100000
    21.220000,    25.100000,    92.100000

```

Now suppose that later in the model we have variables  $x_{dc}$ , representing the flow from depot  $d$  to customer  $c$ . The variables are specified by

```

VARIABLES
  x( ID, IC)

```

and the total cost is expressed as

```

CONSTRAINTS
  TotCost: SUM(d=ID, c=IC) COST(d,c)*x(d,c) $

```

### 5.6.2 Example 2: Index sets in the Coco problem

Now that we have seen a simple introductory example, let us give some more complex examples based on the Coco model from Chapter 4. The notation "**\*n**" in the model listing refers to the notes which follow.

The first Coco example is:

```

! An example of using Index Sets for a simplified Coco
! Model.

LET
  NP=2          ! number of products (p)          (*1
  NF=2          !          factories (f)
  NR=2          !          raw materials (r)
  NT=4          !          time periods (t)

INDICES
  ! Use the names of the factories,
  ! products,
  ! raw materials and time periods

  IProd(NP)
  IRMat(NR)

```

(\*2

---

```
IFact(NF)
ITime(NT+1)
```

```
TABLERMREQ(IProd,IRMat)      ! requirement of product p for raw
                             ! material r
MAXSELLT(IProd,ITime)      ! maximum amount of p that can be
                             ! sold in period t
MAXPROD(IFact)             ! maximum amount fact f can make
SPT(IProd,ITime)           ! selling price of product p in
                             ! period t
RMPT(IRMat,ITime)          ! price of raw mat. r in period t
VC(IProd,IFact)            ! variable cost of product p in
                             ! factory f
INIM(IRMat,IFact)          ! initial raw material r stock
                             ! level in factory f
INIP(IProd,IFact)          ! initial final product p stock
                             ! level in factory f
SET DYNINDEX               ! set up the index names "on the
                             ! fly" (*4
```

```
DISKDATA -s                ! Read data in sparse format (*5
RMREQ      = rmreq.dat
MAXSELLT   = maxsellt.dat
MAXPROD    = maxprod.dat
SPT        = spt.dat
RMPT       = rmpt.dat
VC         = vc.dat
INIM       = inim.dat
INIP       = inip.dat
```

```
VARIABLES
sp(IProd,IFact,ITime)      ! stock level of final product p
                             ! in factory f at beg. of period t
sm(IRMat,IFact,ITime)      ! stock level of raw material r in
                             ! factory f at start of period t
sell(IProd,IFact,ITime)    ! amount of product p sold from
                             ! factory f in period t
manuf(IProd,IFact,ITime)   ! amount of product p made in
                             ! factory f in period t
buy(IRMat,IFact,ITime)     ! amount of raw material r bought
                             ! for factory f in period t
```

```
CONSTRAINTS
max:SUM(p=IProd,f=IFact,t=ITime) SPT(p,t)*sell(p,f,t) &
    -SUM(p=IProd,f=IFact,t=ITime) VC(p,f)*manuf(p,f,t) &
    -SUM(r=IRMat,f=IFact,t=ITime) RMPT(r,t)*buy(r,f,t) &
    -SUM(p=IProd,f=IFact,t=ITime) 2*sp(p,f,t)          &
```

```

-SUM(r=IRMat,f=IFact,t=ITime) 1*sm(r,f,t) $
! final product
SB(p=IProd,f=IFact,t=ITime): &
sp(p,f,t+1) = sp(p,f,t)- sell(p,f,t)+manuf(p,f,t)
! raw material and stock balance
SR(r=IRMat,f=IFact,t=ITime): &
sm(r,f,t+1) = sm(r,f,t)+buy(r,f,t) &
-SUM(p=IProd)RMREQ(p,r)*manuf(p,f,t)
! selling limit
MXS(p=IProd,t=ITime): SUM(f=IFact) sell(p,f,t) < &
MAXSELLT(p,t)
! maximum production
MXM(f=IFact,t=ITime): SUM(p=IProd) manuf(p,f,t) < &
MAXPROD(f)
! raw material storage
SCRM(f=IFact,t=ITime):SUM(r=IRMat) sm(r,f,t) < 300

BOUNDS
sp(p=IProd,f=IFact,1) = INIP(p,f) ! initial product
! levels
sm(r=IRMat,f=IFact,1) = INIM(r,f) ! initial raw material
! levels

BIFGENERATE ! NOTE: we generate a BIF file here (*6
DISKDATA -os ! When we come back we want to see the
! manufacturing plan
manuf.txt = manuf (*7

```

Notes:

1. The LET section as usual sets the parameters.
2. The INDICES section defines and gives the maximum set sizes of the indices that will be used later in the model. At present, the index names are not defined.
3. TABLES are defined, indexed by elements from the index sets.
4. The SET DYNINDEX command tells XPRESS-MP that it is to build up the list of indices that are in the index sets from the data that are read in by subsequent DISKDATA statements.
5. Data are then read into the tables in sparse format. At the same time, the lists of names of indices in the index sets are created. For instance, the file rmreq.dat contains data:

```

"Product1", "Rawmat_1", 1.000000
"Product1", "Rawmat_2", .500000
"Product2", "Rawmat_1", 1.300000
"Product2", "Rawmat_2", .400000

```

---

so after it has been read the index set IProd contains indices:

```
"Product1"  
"Product2"
```

and the index set IRMat contains indices:

```
"Rawmat_1"  
"Rawmat_2"
```

6. The BIFGENERATE command is crucial here, since it generates a binary matrix file (a .bif file) in which variables and constraints are labelled by a unique *sequence number* rather than eight character MPS names.

The GENERATE command which we have been using up to now generates an MPS matrix file in which the variables and constraints are labelled using eight character MPS names (see also **3.2.3 Obtaining the Solution**). The MPS names are derived from the generic variable or constraint name and two characters from the index set elements. As the index set elements differ only at the eighth character, the MPS names will not be distinct. For example, here is an extract of the MPS matrix file which shows the MPS names generated:

```
NAME COCO_IS  
ROWS  
N max  
E SBPrFaPe  
E SBPrFaPe  
...  
E SBPrFaPe  
...  
COLUMNS  
spPrFaPe max -2.000000 SBPrFaPe -1.000000  
spPrFaPe max -2.000000 SBPrFaPe -1.000000  
...
```

Trying to read this into the optimiser would result in many "?58 Duplicate element" or "?97 split vector" error messages. It is possible to change the way MPS names are generated but as at least eight characters from the index set elements would be required to distinguish them there is not a lot that can be done. See the **VARIABLES Section** and the **INDICES Section** in chapter 4, *The mp-model Model Builder*, of the *XPRESS-MP Reference Manual* for more details.

However, no such problems occur using BIFGENERATE and binary matrix files, as the names are not important. Binary matrix files are input into the optimiser using DASHIN instead of INPUT.

7. On re-invoking the XPRESS-MP modeller and restoring the solution we get the optimal values of the manuf variables into the file manuf.txt.

---

There is something special to note about the time period indices in this model. If we inspect the SB constraint we see a reference to  $t+1$ . The dynamic initialisation of the underlying set `Itime` has only set up 4 values since the file `maxsellt.dat` holds:

```
"Product1", "Period_1", 650.000000
"Product1", "Period_2", 600.000000
"Product1", "Period_3", 500.000000
"Product1", "Period_4", 400.000000
"Product2", "Period_1", 600.000000
"Product2", "Period_2", 500.000000
"Product2", "Period_3", 300.000000
"Product2", "Period_4", 250.000000
```

The dummy subscript  $t$  will go through these names, and it is reasonable to expect that for  $t = \text{Period\_1}$   $t+1$  will be `Period_2`. since the latter name was seen immediately after the former. But what is to be done with the successor to `Period_4`? As there was no fifth index but the index set was of size 5 the fifth index name is null. If we are using `BIFGENERATE` and do not need ever to know MPS names, then this is as good a name as any.

### 5.6.3 Example 3: Mixing index sets and numerical indices

The second Coco example illustrates:

- mixing index set and conventional integer subscripts; and
- generating legal MPS with index sets.

```
! An example of using Index Sets for a simplified Coco
! Model, but having time indexed by integers.
! The data files have suffix .dau
```

```
LET
```

```
NP=2
NF=2
NR=2
NT=4
```

```
INDICES      ! We use the names of the factories,
IProd(NP)    ! products, and raw materials only.
IRMat(NR)
IFact(NF)
```

```
TABLES
```

```
RMREQ(IProd,IRMat)
MAXSELLT(IProd,NT)
MAXPROD(IFact)
SPT(IProd,NT)
RMPT(IRMat,NT)
VC(IProd,IFact)
```

---

```

INIM(IRMat,IFact)
INIP(IProd,IFact)

SET DYNINDEX

DISKDATA -s          ! Read data in sparse format again
RMREQ      = rmreq.dau ! but from slightly different files
MAXSELLT   = maxsellt.dau
MAXPROD    = maxprod.dau
SPT        = spt.dau
RMPT       = rmpt.dau
VC         = vc.dau
INIM       = inim.dau
INIP       = inip.dau

VARIABLES
sp(IProd,IFact,NT+1) ! Note that NT is used here
sm(IRMat,IFact,NT+1)
sell(IProd,IFact,NT)
manuf(IProd,IFact,NT)
buy(IRMat,IFact,NT)

CONSTRAINTS
max:
    SUM(p=IProd,f=IFact,t=1:NT) SPT(p,t)*sell(p,f,t) &
    ! not t=ITime
-SUM(p=IProd,f=IFact,t=1:NT) VC(p,f)*manuf(p,f,t) &
-SUM(r=IRMat,f=IFact,t=1:NT) RMPT(r,t)*buy(r,f,t) &
-SUM(p=IProd,f=IFact,t=1:NT) 2*sp(p,f,t) &
-SUM(r=IRMat,f=IFact,t=1:NT) 1*sm(r,f,t) $

SB(p=IProd,f=IFact,t=1:NT): &
sp(p,f,t+1) = sp(p,f,t)-sell(p,f,t)+manuf(p,f,t)

SR(r=IRMat,f=IFact,t=1:NT): &
sm(r,f,t+1) = sm(r,f,t)+buy(r,f,t) &
-SUM(p=IProd)RMREQ(p,r)*manuf(p,f,t)

MXS(p=IProd,t=1:NT): &
SUM(f=IFact)sell(p,f,t) < MAXSELLT(p,t)

MXM(f=IFact,t=1:NT): &
SUM(p=IProd)manuf(p,f,t) < MAXPROD(f)

SCRM(f=IFact,t=1:NT):SUM(r=IRMat) sm(r,f,t) < 300

BOUNDS
sp(p=IProd,f=IFact,1) = INIP(p,f)
sm(r=IRMat,f=IFact,1) = INIM(r,f)

GENERATE          ! Note - we generate MPS here

```

---

```
DISKDATA -os
manuf.txt = manuf
```

The original comments have been removed, and the new comments refer to the difference between this model and the first one. Note that time has been treated as an integer, so given  $t$  there is no doubt about the meaning of  $t+1$ . This time the file from which we get the data for the MAXSELLT table is:

```
"p1", 1, 650.000000
"p1", 2, 600.000000
"p1", 3, 500.000000
"p1", 4, 400.000000
"p2", 1, 600.000000
"p2", 2, 500.000000
"p2", 3, 300.000000
"p2", 4, 250.000000
```

so the index set  $I_{Prod}$  has members "p1" and "p2". The second subscript is indexed by integers.

With this model we can generate a legal MPS matrix file since the index names are unique in the first two characters. We can see this by looking at part of the MPS matrix file:

```
NAME          COCO_ISX
ROWS
  N  max_____
  E  SB_p1f11
  E  SB_p1f12
  ...
  E  SR_r1f11
  E  SR_r1f12
  ...
  E  SR_r2f11
  E  SR_r2f12
  E  SR_r2f13
  E  SR_r2f14
  E  SR_r2f21
  E  SR_r2f22
  E  SR_r2f23
  E  SR_r2f24
  ...
COLUMNS
  sp_p1f11  max_____  -2.000000  SB_p1f11  -1.000000
  sp_p2f11  max_____  -2.000000  SB_p2f11  -1.000000
  ...
RHS
  RHS00001  MXS_p101  650.000000  MXS_p102  600.000000
  RHS00001  MXS_p103  500.000000  MXS_p104  400.000000
  ...
```

---

Notice that for the SB constraints the  $p$  and  $f$  subscripts are encoded with the index names, whilst the  $t$  subscript is encoded with the integer value of  $t$ .

## 5.7 Built-In Functions

There is a range of built-in functions available in the XPRESS-MP modelling language. They are described fully in the *Arithmetic and Logical Functions* section of **Chapter 4** in the *XPRESS-MP Reference Manual*.

### Financial

$\text{disc}(a, t)$ , defined by  $\text{disc}(a, t) = \frac{1}{(1+a)^{t-1}}$  and for use in calculating discount factors. For example, an item valued at \$100.00 discounted at 5% per annum over 3 years would give  $a=0.05$  and  $t=3$   $\text{disc}(0.05, 3)$  gives 0.907029. Hence at the end of the 3 year period, the original item would be worth  $\$100.00 \times 0.907029 = \$90.70$ .

### Exponentiation/logarithms

$\exp(a)$ , the natural exponent: i.e.,  $\exp(a) = e^a$ .

$\ln(a)$ , the natural logarithm of  $a$ . Its argument must be greater than zero.

### Rounding

$\text{abs}(a)$ , the absolute value of  $a$ .  $\text{abs}(a) = |a|$

$\text{ceil}(a)$ , the smallest integer not less than its argument.

$\text{floor}(a)$ , the largest integer not greater than the given argument.

$\text{int}(a)$ , the truncated value of a real argument. For example,  $\text{int}(4.7)$  gives 4.0; and  $\text{int}(-3.375)$  gives -3.0.

$\text{mod}(a, b)$ , the modulus (or remainder) defined by:  $\text{mod}(a, b) = a - \text{int}(a/b) * b$  i.e., the remainder when  $a$  is divided by  $b$ , an integer number of times. Note this definition also holds true if either or both arguments are real.

$\text{round}(a)$ , the nearest integer to its real argument. For example,  $\text{round}(4.7)$  gives 5.0; and  $\text{round}(-3.375)$  gives -3.0. Compare this with  $\text{int}()$  which provides truncation.

### Maxima and minima

$\text{max}(a1, a2, \dots)$ , the largest of the given arguments.

$\text{min}(a1, a2, \dots)$ , the smallest of the given arguments.

### Post-processing

---

The use of these functions is beyond the scope of this User Guide. They are listed here for completeness, but please refer to **4.3 The Building Blocks of mp-model** in the *XPRESS-MP Reference Manual* for details. See also a brief example in **5.8.1 Advanced Techniques**.

`dj(x)`, reduced cost, `dj`. This is for use after optimisation and allows easy access to the reduced costs of decision variables.

`dual(c)`, dual value. This is for use after optimisation and allows easy access to the dual values of constraints.

`slack(c)`, slack value. This is for use after optimisation and allows easy access to the slack values of constraints.

#### Data input/output

`records`, the number of records read or written by the last `DISKDATA` command. No arguments and hence no parentheses are required.

#### Error functions

`errors`, the number of errors encountered so far within the modeller. No arguments and hence no parentheses are required.

`errorno`, a number indicating the error status of the last command. A zero value indicates no error and a non-zero value indicates an error. No arguments and hence no parentheses are required.

## 5.8 Exporting Results

At a first reading, many users of XPRESS-MP do not see **4.6 Basic Report Writing with mp-model** in the *XPRESS-MP Reference Manual* yet this describes the most powerful way of exporting the solution ready for external interpretation or application. You can export the solution values of key model variables directly to text files, databases or spreadsheets using similar notation that you use to read data in from those sources. In this section we show the basic functionality.

The method is very simple: by including statements in the model after the `GENERATE` (or `END` or `BIFGENERATE`) statement, you can refer to variables as if they were tables containing the optimal solution values. You can export the solution values of certain key variables, or even define new tables and use `ASSIGN` statements to manipulate the solution before exporting it in a more useful form. Once the problem has been solved these statements are executed.

Let's illustrate this with a small example. Suppose we have the model `knap`.

```
LET NItems=8 ! Number of items
```

```
TABLES
```

---

```

V(NItems) ! Value of items
W(NItems) ! Weight of items

DATA
!Item 1  2  3  4  5  6  7  8
V = 15,100, 90, 60, 40, 15, 10, 1
W = 2, 20, 20, 30, 40, 30, 60, 10

LET WTMAX = 102

VARIABLES
x(NItems) ! 1 if we take item n; 0 otherwise

CONSTRAINTS
ValMax: SUM(i=1:NItems)V(i)*x(i) $ ! maximise the value
Wmax: SUM(i=1:NItems)W(i)*x(i)< WTMAX ! max weight

BOUNDS
x(i=1:NItems) .BV. ! All x are 0/1

END

DISKDATA -o
xopt.dat = x
valmax.dat = ValMax

```



To use mp-model and mp-opt in the conventional way, we might say

```

mp-model knap
INPUT

```

to parse the model and generate the matrix, and then

```

mp-opt knap
INPUT
MAXIMISE
GLOBAL
QUIT

```

to solve the problem.

If we now restart mp-model and type

```

mp-model knap
RESTORE
INPUT
QUIT

```

---

the commands after the END statement will be obeyed.

Let us examine this final step in more detail. The RESTORE statement restores the previous state of mp-model when the END statement was reached by loading a model save file, knap.svm. This file is generated automatically by mp-model upon matrix generation if statements are detected after the matrix generation command. The INPUT command instructs mp-model to read in the remaining statements from the model file.



In **Visual XPRESS**, solve the problem in the usual way by selecting Run > Solve LP and then Run > Solve Global. Then, to execute the remaining model statements, select Run > Post Optimal Processing.

When the remaining model statements

```
DISKDATA -o
  xopt.dat    = x
  valmax.dat  = ValMax
```

are executed, the optimal values of the variables x and the objective function ValMax are written out to text files xopt.dat and valmax.dat respectively. We've already seen (in 5.5 *Displaying Data*) how DISKDATA -o statements write a data table out to a text file. Here we see how we can write out the values of variables and the objective function once the problem has been solved.

Note that it is sometimes helpful to include a full path in the filenames in DISKDATA -o statements if you want to write files in an explicit location.

We can use the post-processing facility to output to spreadsheets and databases too using DISKDATA -co to combine data output with a database connection. See chapter 5, *Extending mp-model Using XPRESS-Connect*, in the *XPRESS-MP Reference Manual* for details and examples.

Just to give you an idea, here is a simple model extract to export the solution of the knap model to an Excel spreadsheet using an ODBC connection.

```
! See above for first part of knap model
END

CONNECT odbc, 'DSN=Excel Files; DBQ=c:\models\knap.xls'
DISKDATA -oc
  'select * from XOPT'    = x
  'select * from VALMAX'  = ValMax
DISCONNECT
```

The CONNECT statement connects to the specified spreadsheet c:\models\knap.xls. The DISKDATA statements then write out the optimal values of the variables x and the objective function ValMax to Excel ranges XOPT and

---

VALMAX respectively. These ranges must have already been initialised in the spreadsheet – see the subsection *Data Output* in 5.2 *The ODBC Interface*, of the *XPRESS-MP Reference Manual*.

### 5.8.1 Advanced Techniques

There are many enhancements to the DISKDATA output section that allow very powerful solution export. Here we just skim the surface of what's possible.

Using the modeller functions `dj`, `slack` and `dual` you can output the reduced costs of variables and the slack and dual values of constraints, e.g:

```
DISKDATA -o
xdj.dat      = dj(x)
wmaxsl.dat   = slack(Wmax)
```

although of course the reduced costs and dual values only make sense for linear problems.

Using the DISKDATA sparse option `-s` we can output the values in sparse format, i.e., the values of a model entity `x(index1, index2, ...)` are output in the form `index1, index2, ..., value`.

```
DISKDATA -so
xopt.dat    = x
```

If you wish to prevent zero values being output, you must define the variable `x` to be 'sparse' thus:

```
VARIABLES
x(NItems) -e      ! 1 if we take item n; 0 otherwise
```

The `-e` option for variables is used by analogy with sparse data tables – it has no effect other than to prevent zero values being output in a DISKDATA section.

## 5.9 Case Management

If the problem name specified on starting up includes an absolute or relative path then when `mp-model` cannot find an input file - model file or data file - it will search preceding directories in the path if the `SET UPDIR` command has been issued.

For example if `mp-model` was run using the problem name `\problem\case1\sub2` then if an input file cannot be found in the directory `\problem\case1`, `mp-model` will look in `\problem` and `\` for the required input file.

If `mp-model` is run with a relative path and problem name such as `..\problem\case1\sub2`, `mp-model` will search for input files in the directories

---

..\problem\case1, ..\problem, .. and finally the directory from which mp-model was called up.

Note that on UNIX™ platforms the directory separator is a "/" not a "\".

## 5.10 Macros

Macros are a very powerful part of XPRESS-MP. We shall illustrate some examples of their use here.

### Example 1: Union of two index sets

It can sometimes be useful to create the union of two index sets. A macro to do this is as follows:

```
MACRO cre_union a,b,c
INDICES c(entries(a)+entries(b))
ASSIGN
  c(i=1:entries(a))=a(i)
  FOR(i=1:entries(b)|
    .not.belongs(b(i),a)):c(entries(c)+1) = b(i)      &
ENDMACRO
```

The index set *c* is created and assigned values of the union of index sets *a* and *b*. Thus

```
INDICES
  cities(6)
  ports(10)
DATA
  cities="rome","bristol","london","paris","lpool"
  ports ="plymouth","bristol","glasgow","london",      &
    "calais","lpool"
  cre_union cities,ports,places
```

creates a new index set, *places*, whose elements are:

```
"rome", "bristol", "london", "paris", "lpool", "plymouth",
"glasgow", and "calais".
```

### Example 2: Intersection of two index sets

Similarly, the following macro will create an index set, *c*, which is the intersection of index sets *a* and *b*:

```
MACRO cre_inter a,b,c
INDICES c(min(entries(a), entries(b)))
ASSIGN FOR(i=1:entries(a)|belongs(a(i),b)):      &
  c(entries(c)+1) = a(i)
ENDMACRO
```

---

Thus

```
cre_inter cities,ports,both
```

creates a new index set, both, whose elements are:

```
"bristol", "london" and "lpool"
```

### Example 3: Blending constraints

The blending constraints that restrict the qualities of a blended product are given by the inequalities:

$$\sum_j (Q_{ij} - L_i) x_j \geq 0 \quad \forall i \quad \text{and} \quad \sum_j (Q_{ij} - U_i) x_j \leq 0 \quad \forall i$$

where:  $x_j$  is the quantity of material  $j$  blended; the  $i$ th quality of material  $j$  is  $Q_{ij}$ ; and that quality has lower and upper limits  $L_i$  and  $U_i$ .

```
MACRO QUALITY_LIMITS x, Num_mats, Quality, Lower, Upper
CONSTRAINTS
    L x(i=1:entries(Lower)):
        sum(j=1:Num_mats) (Quality(i,j)-Lower(i))*x(j) > 0
CONSTRAINTS
    U x(i=1:entries(Upper)):
        sum(j=1:Num_mats) (Quality(i,j)-Upper(i))*x(j) < 0
ENDMACRO

VARIABLES
    x(3)

TABLES
    QUAL(2,3)
    LOW(2)
    UPP(2)

DATA
    QUAL(1,1) = 0.3, 0.4, 0.5
    QUAL(2,1) = 0.4, 0.5, 0.6
    LOW = 0.4, 0.5
    UPP = 0.5, 0.6

QUALITY_LIMITS x, 2, QUAL, LOW, UPP
```

This will generate the two sets of constraints:

```
Lx(i=1:2): SUM(j=1:2) (QUAL(i,j)-LOW(i))*x(j) > 0
Ux(i=1:2): SUM(j=1:2) (QUAL(i,j)-UPP(i))*x(j) < 0
```



---

## 6 Integer Programming in XPRESS-MP

### 6.1 Introduction to Integer Programming

Though many systems can accurately be modelled as Linear Programs, there are situations where discontinuities are at the very core of the decision making problem. There seem to be three major areas where non-linear facilities are required

- where entities must inherently be selected from a discrete set;
- in modelling logical conditions; and
- in finding the global optimum over functions.

The XPRESS-MP modelling language lets you model these non-linearities using a range of *discrete (global) entities* and then the Mixed Integer Programming (MIP) optimiser can be used to find the overall (global) optimum of the problem. Usually the underlying structure is that of a Linear Program, but optimisation may be used successfully when the non-linearities are separable into functions of just a few variables.

XPRESS-MP handles the following global entities:

**Binary variables (BV)** - decision variables that can take either the value 0 or the value 1 (do/don't do variables).

**Integer variables (UI)** - decision variables that can take only integer values. Some small upper limit must be specified.

**Partial integer variables (PI)** - decision variables that can take integer values up to a specified limit and any value above that limit.

**Semi-continuous variables (SC)** - decision variables that can take either the value 0, or a value between some lower limit and upper limit. SCs help model situations where if a variable is to be used at all, it has to be used at some minimum level.

**Special Ordered Sets of type one (SOS1 or S1)** - an ordered set of variables at most one of which can take a non-zero value.

**Special Ordered Sets of type two (SOS2 or S2)** - an ordered set of variables, of which at most two can be non-zero, and if two are non-zero these must be consecutive in their ordering.

---

The most commonly used entities are binary variables, which can be employed to model a whole range of logical conditions. General integers are more frequently found where the underlying decision variable really has to take on a whole number value for the optimal solution to make sense. For instance, we might be considering the number of aeroplanes to charter, where fractions of an aeroplane are not meaningful and the optimal answer will probably involve so few planes that rounding to the nearest integer may not be satisfactory.

Partial integers provide some computational advantages in problems where it is acceptable to round the LP solution to an integer if the optimal value of a decision variable is quite large, but unacceptable if it is small.

Semi-continuous variables are useful where, if some variable is to be used, its value must be no less than some minimum amount.

Special Ordered Sets of type 1 are often used in modelling choice problems, where we have to select at most one thing from a set of items. The choice may be from such sets as: the time period in which to start a job; one of a finite set of possible sizes for building a factory; which machine type to process a part on.

Special Ordered Sets of type 2 are typically used to model non-linear functions of a variable. They are the natural extension of the concepts of Separable Programming, but when embedded in a Branch and Bound code (see below) enable truly global optima to be found, and not just local optima. (A local optimum is a point where all the nearest neighbours are worse than it, but where we have no guarantee that there is not a better point some way away. A global optimum is a point which we know to be the best. In the Himalayas the summit of K2 is a local maximum height, whereas the summit of Everest is the global maximum height).

Theoretically, models that can be built with any of the entities we have listed above can equally well be modelled solely with binary variables. The reason why modern IP systems have some or all of the extra entities is that they often provide significant computational savings in computer time and storage when trying to solve the resulting model. Most books and courses on Integer Programming do not emphasise this point adequately. We have found that careful use of the non-binary global entities often yields very considerable reductions in solution times over ones that just use binary variables.

The Integer Programming extension to XPRESS-MP uses a *Branch and Bound* search to locate and prove the optimal solution. It has a set of default strategies which have been found to work well on most problems. However, the user should note carefully that sophistication in modelling is important in large scale MIP work. Theoretical developments in the last decade have led to insights as to how best to model IP problems, and how to choose between alternative formulations. We cannot over-stress the point that a good formulation can often speed an IP search by orders of magnitude. Dash are always interested in discussing IP formulations with XPRESS-MP users.

---

The special features of the integer programming extension to XPRESS-MP are called up after solving the underlying Linear Programming problem to optimality (using MAXIMISE or MINIMISE). Then the non-linearities and discreteness that have been specified in the INPUT data are recognised. The GLOBAL command starts the search for the optimal integer solution.



A minimal run stream to solve an integer programming problem might be:

```
INPUT
MINIMISE
GLOBAL
PRINT
QUIT
```

Note that if you are not interested in the solution to the LP relaxation, you can solve the IP in one go using MINIM -g instead of MINIMISE followed by GLOBAL. This is also slightly more efficient.



Click Run Solve LP to solve the underlying linear program, and then click Run Solve Global to solve the IP. As usual, Run Solution Print File will display the solution, although this is not recommended for large scale problems.

To illustrate the use of XPRESS-MP in modelling Integer Programming problems, a small example follows. The first formulation uses binary variables. This formulation is then modified to show you how to use Special Ordered Sets.

For the interested reader, an excellent text on Integer Programming is *Integer Programming* by Laurence Wolsey, Wiley Interscience, 1998, ISBN 0-471-28366-5.

## 6.2 A Project Planning Model

The problem to be modelled is as follows:

A company has several projects that it must undertake in the next few months. Each project lasts for a given time (its duration) and uses up one resource as soon as it starts. The resource profile is the amount of the resource that is used in the months following the start of the project. For instance, project 1 uses up 3 units of resource in the month it starts, 4 units in its second month, and 2 units in its last month.

The problem is to decide when to start each project, subject to not using more of any resource in a given month than is available. The benefit from the project only starts to accrue when the project has been completed, and then it accrues at  $BEN_p$  per month for project  $p$ , up to the end of the time horizon.

Below, we give a mathematical formulation of the above project planning problem, and then display the XPRESS-MP model form.

We define the following constants:

$NPROJ$ : the number of projects; and  
 $NMTH$ : the number of months to plan for.

The data are:

$PROF_{pt}$ : the resource usage of project  $p$  in its  $t^{\text{th}}$  month  
 $BEN_p$ : the benefit per month when project finishes  
 $RES_m$ : the resource available in month  $m$   
 $DUR_p$ : the duration of project  $p$

and the variables:

$x_{pm}$ : =1 if project  $p$  starts in month  $m$ , otherwise 0  
 $start_p$ : start month for project  $p$

The objective function is obtained by noting that the benefit coming from a project only starts to accrue when the project has finished. If it starts in month  $m$  then it finishes in month  $m+DUR_p-1$ . So, in total, we get the benefit of  $BEN_p$  for  $NMTH-(m+DUR_p-1) = NMTH - m - DUR_p + 1$  months. We must consider all the possible projects, and all the starting months that let the project finish before the end of the planning period. For the project to complete it must start no later than month  $NMTH-DUR_p$ . Thus the profit is:

$$profit = \sum_{p=1}^{NPROJ} \sum_{m=1}^{NMTH-DUR_p} BEN_p (NMTH - m - DUR_p + 1) x_{pm}$$

Each project must be done once, so it must start in one of the months 1 to  $NMTH-DUR_p$ :

$$\sum_{m=1}^{NMTH-DUR_p} x_{pm} = 1 \quad \forall p$$

We next need to consider the implications of the limited resource availability each month. Note that if a project  $p$  starts in month  $m$  it is in its  $(k-m+1)^{\text{th}}$  month in month  $k$ , and so will be using  $PROF_{p,k-m+1}$  units of the resource. Adding this up for all projects and all starting months up to and including the particular month  $k$  under consideration gives:

$$\sum_{p=1}^{NPROJ} \sum_{m=1}^k PROF_{p,k+1-m} x_{pm} \leq RES_k \quad \forall k$$

---

The start month of a project is given by:

$$\sum_{m=1}^{NMTH-DUR_p} mx_{pm} = start_p \quad \forall p$$

since if an  $x_{pm}$  is 1 the summation picks up the corresponding  $m$ .

Finally we have to specify that the  $x_{pm}$  are binary (0 or 1) variables. This is done by the statement:

$$x_{pm} \in \{0,1\} \quad \forall p, m$$

The model as specified to XPRESS-MP is as follows:

```
MODEL IP example
  LET NPROJ=3          ! Number of projects
  LET NMTH=6           ! Number of months to plan for

VARIABLES
  x(NPROJ,NMTH)        ! x(p,m)=1 if proj p starts in month m
  start(NPROJ)         ! start month for project p

TABLES
  PROF(NPROJ,NMTH)     ! Resource usage of project p in its t-th
                        ! month
  BEN(NPROJ)           ! Benefit per month when project finishes
  RES(NMTH)            ! Resource available in month m
  DUR(NPROJ)           ! Duration of project p

DATA
  DUR(1) = 3, 3, 4
  PROF(1,1) = 3, 4, 2
  PROF(2,1) = 4, 1, 6
  PROF(3,1) = 3, 2, 1, 2
      ! Other PROF entries are 0 by default
  BEN(1) = 10.2, 12.3, 11.2
  RES(1) = 5, 6, 5, 5, 4, 5 ! Resource avail in month m

CONSTRAINTS

! The objective function.
OBJFmax: SUM(p=1:NPROJ,m=1:NMTH-DUR(p))    &
          (BEN(p)*(NMTH-m-DUR(p)+1)) * x(p,m)    $

! We have to do each project, but only once.
ONEX(p=1:NPROJ): SUM(m=1:NMTH-DUR(P)) x(p,m) = 1.0

! If a project starts at time m it is in its
! k-m+1th month in month k
```

---

```

MAXB(k=1:NMTH):
    SUM(p=1:NPROJ,m=1:k) PROF(p,k+1-m)*x(p,m) < RES(k)

! start(p) gets value of start month for project p
RR(p=1:NPROJ): SUM(m=1:NMTH-DUR(P)) m*x(p,m) = start(p)

BOUNDS
    x(p=1:NPROJ,m=1:NMTH) .BV.

GENERATE

```

### 6.3 A Project Planning Model Using Special Ordered Sets

The example can be modified to use Special Ordered Sets of type 1 (SOS1). The  $x_{pm}$  variables for a given  $p$  form a set of variables which are ordered by  $m$ , the month. The ordering is created by the coefficients of the  $x(p, m)$  in the RR rows. For example,  $x_{p1}$ 's coefficient, 1, is less than  $x_{p2}$ 's, 2, which in turn is less than  $x_{p3}$ 's coefficient, and so on. The row which induces the ordering for the set is called the set's *reference row*.

The fact that the  $x_{pm}$  variables for a given  $p$  form a set of variables and that the reference row is RR( $p$ ) is specified to the modeller as follows:

```

SETS
    SS(p=1:NPROJ): SUM(m=1:NMTH) x(p,m) .S1. RR(p)

```

The SETS line tells XPRESS-MP that special ordered set specifications follow. The name of the sets is SS, and one separate set is to be created for each value of  $p$ . So far this is just like a constraint specification. Now comes something even more like a component of a constraint, but which is really an abuse of a mathematical term. The SUM term is not a summation operator but really XPRESS-MP's equivalent to the set theoretic union concept. In this context it says that all the  $x(p, m)$  variables from  $m=1$  to NMTH are to be considered members of an SOS1 (because of the .S1. term), with a reference row RR( $p$ ). If the set were an SOS2 set then the .S1. term would be replaced by .S2..

The specification of the  $x(p, m)$  as binary variables must now be removed. The binary nature of the  $x(p, m)$  is implied by the SOS1 property, since if the  $x(p, m)$  must add up to 1 and only one of them can differ from zero, then just one is 1 and the others are 0.

If the two formulations are equivalent why were Special Ordered Sets invented, and why are they useful? The answer lies in the way the reference row gives the search procedure in Integer Programming (IP) good clues as to where the best solution lies. Quite frequently the Linear Program (LP) that is solved as a first approximation to an Integer Program gives an answer where  $x_{pI}$  is fractional, say with a value of 0.5, and  $x_{pM}$  takes on the same fractional value. The IP will say:

"my job is to get variables to 0 or 1. Most of the variables are already there so I will try moving  $x_{pI}$  or  $x_{pM}$ . Since the set members must add up to 1.0, one of them will go

---

to 1, and one to 0. So I think that we start the project either in the first month or in the last month."

A much better guess is to see that the  $x_{pm}$  are ordered and the LP is telling us it looks as if the best month to start is somewhere midway between the first and the last month. When sets are present, the IP can branch on sets of variables. It might well separate the months into those before the middle of the period, and those later. It can then try forcing all the early  $x_{pm}$  to 0, and restricting the choice of the one  $x_{pm}$  that can be 1 to the later  $x_{pm}$ . It has this option because it now has the information to "know" what is an early and what is a late  $x_{pm}$ , whereas these variables were unordered in the binary formulation.

The power of the set formulation can only really be judged by its effectiveness in solving large, difficult problems. When it is incorporated into a good IP system such as XPRESS-MP it is often found to be an order of magnitude better than the equivalent binary formulation for large problems.



---

# Glossary of Terms

## **binary variable**

A decision variable that may take the values 0 or 1 in a solution. Problems that contain binary variables are mixed integer programming problems.

## **bound**

A simple constraint, used to set simple upper or lower bounds on a decision variable, or to fix a decision variable to a given value. In mixed integer programming (MIP) problems, used to define variables as binary or integer variables, etc.

## **command (in the modeller)**

A single line section in an XPRESS-MP model, such as `GENERATE` or `INPUT` is sometimes called a command.

## **constant**

An object whose value is known. Constants are also referred to as data parameters. Sometimes data refers to the numerical coefficients in constraints, whereas parameters refers to the dimension and number of the objects in a model.

## **constraint**

A linear inequality (or equation) that must be satisfied by the value of the decision variables at the optimal solution.

## **constraint type**

There are four types of constraints:  $\geq$  (greater than or equal to),  $\leq$  (less than or equal to),  $=$  (equal to) or \$ (unconstraining, e.g., an objective function).

## **continuous variable**

A decision variable that may take any value between 0 and infinity (or between some lower and upper bound in general). Also called a linear variable.

---

## **data**

An object whose value is known. Constants are also referred to as data or parameters. Sometimes data refers to the numerical coefficients in the constraints, whereas parameters refers to the dimension and number of the objects in a model.

## **data table**

An array of data elements (constants) to be used in a model.

## **decision variable**

An object (an 'unknown') whose value is to be determined by optimisation. In linear programming (LP) problems, all decision variables are linear (or continuous), that is, they may take any value between 0 and infinity (or between a lower and upper bound in general). In mixed integer programming (MIP) problems, some variables may be binary, integer etc., or form a special ordered set. Sometimes referred to simply as *variables*.

## **dj**

See reduced cost.

## **dual value**

The change in the objective value per unit change in the RHS of a constraint. Sometimes referred to as the shadow price – the “price” of the resource being rationed by the constraint.

## **global entity**

Any integer, binary, partial integer or semi-continuous variable or a special ordered set.

## **input cost**

The original objective coefficient of a variable.

## **integer programming (IP) problem**

An alternative name for a mixed integer programming (MIP) problem, in particular, one which doesn't contain any linear variables.

## **integer variable**

A decision variable that may take the values 0, 1, 2, ... up to some small upper limit in a solution. Problems that contain integer variables are mixed integer programming problems.

---

## **keyword (in the modeller)**

A keyword in an XPRESS-MP model introduces a new section. Typical keywords are VARIABLES and CONSTRAINTS.

## **linear expression or function**

A term (or sum of terms) of the form  $\pm \text{constant} \times \text{decision variable}$ .

## **linear inequality or equation**

A linear expression that must be greater-than-or-equal-to, less-than-or-equal-to, or equal-to a constant term (the right hand side).

## **linear programming (LP) problem**

A problem made up of linear decision variables, an objective function and constraints. The objective function and constraints are linear functions of the decision variables.

## **linear variable**

A decision variable that may take any value between 0 and infinity (or between some lower and upper bound in general). Also called a continuous variable.

## **mixed integer programming (MIP) problem**

A linear programming (LP) problem which contains some global entities, e.g., some of the variables may be binary variables, integer variables, etc, or form a special ordered set.

## **model**

The set of decision variables, objective function, constraints and data that define the problem. Often used to mean the XPRESS-MP representation of the problem.

## **model instance**

A model structure complete with explicit data, i.e., values for the data and parameters.

## **model structure**

The definition of the decision variables, data tables and constraints, and the algebraic relationship between them, without specifying any explicit data or parameters.

## **objective function**

A linear expression which to be optimised (maximised or minimised) by choosing values for the decision variables.

---

### **optimal solution**

A solution that achieves the best possible (maximum or minimum) value of the objective function.

### **parameter**

An object whose value is known. Constants are also referred to as data or parameters. Sometimes data refers to the numerical coefficients in the constraints, whereas parameters refers to the dimension and number of the objects in a model.

### **partial integer variable**

A decision variable that, in a solution, may take the values 0, 1, 2, ... up to some small upper limit, and then any continuous value over that limit. Problems that contain partial integer variables are mixed integer programming problems.

### **reduced cost**

The amount by which the objective coefficient of a decision variable would have to change for the variable to become “active”, i.e., non-zero. Sometimes referred to as the  $d_j$ .

### **right hand side (RHS)**

The constant term in a constraint. By convention the value is written on the right hand side of the constraint, although this is not necessary in XPRESS-MP.

### **section (in the modeller)**

Models in XPRESS-MP are divided into sections, headed by a keyword. Typical sections are the VARIABLES section, which defines the decision variables, and the CONSTRAINTS section, which defines the constraints.

### **semi-continuous variable**

A decision variable that, in a solution, may take the value 0 or any continuous value in a range from a lower limit to an upper limit. Problems that contain semi-continuous variables are mixed integer programming problems.

### **shadow price**

See dual value.

### **slack value**

The amount by which a constraint differs from its RHS.

---

**solution**

A set of values for the decision variables that satisfy the constraints. See also *optimal solution*.

**Special Ordered Set (SOS)**

An ordered set of variables that must fulfill a special condition. In a Special Ordered Set of type 1 (an "SOS1" or "S1 set") at most one variable may be non-zero. In a Special Ordered Set of type 2 (an "SOS2" or "S2 set") at most two variables may be non-zero, and if two variables are non-zero, they must be next to each other in the ordering. In XPRESS-MP the ordering is supplied in a "reference row", which may be an ordinary constraint or a separate unconstraining constraint. There is no requirement that the variables must take integer values.

**variable**

See decision variable.



---

# Index

---

## A

ASSIGN section · 38, 58

---

## B

bif file · see *binary matrix file*

BIFGENERATE · 67, 72

binary matrix file · 67

binary variables · 47, 79, 83

blending problem · 22

bounds · 44

    binary variables · 47

    conditional · 51

    integer variables · 47

BOUNDS section · 44, 51

branch and bound search · 80

---

## C

calculating data · 37, 58

case management · 75

case sensitive · 11

chess set problem · 7, 11

    solution · 13

coco problem

    index sets · 64, 68

    phase 1 · 28, 30, 31

    phase 2 · 34, 38

    phase 3 · 39, 44

    phase 4 · 46

    phase 5 · 48

columns · see *variables*

columns section · 32

comments · 24

    in data files · 37

conditional generation · 51

CONNECT statement · 56, 74

Console XPRESS · 3

constraint name · 10

constraint type · 33

constraints · 8, 10

    arrays of · 35

    conditional · 52

    modelling blend quality · 23, 77

    modelling fixed costs · 47

    modelling logical conditions · 48, 79

CONSTRAINTS section · 9, 10, 36

continuation character · 38

conventions · 2

---

## D

DASHIN · 67

data · 19

    calculating · 37, 58

    dense · 60

    displaying · 60

    exporting · 62

    importing · 22, 37, 53, 54

    ODBC · 55, 74

    saving to a file · 62

    sparse · 59, 62

    text files · 22, 25, 55

data files · 24, 55

    comments in · 37

    dense format · 60

    importing · 25

    sparse format · 59, 62

---

DATA section · 19, 53  
data tables · 17, 18  
    dense · 60  
    sizing from spreadsheet data · 57  
    sparse · 59, 62, 63  
database connection · 55, 74  
decision variables · 8. *see also*  
    *variables*  
dense data tables · 60  
DISCONNECT statement · 56  
discrete entities  
    *see* global entities · 79  
discrete set · 79  
DISKDATA section · 25, 37, 54  
    -c option · 56, 74  
    -d option · 60  
    -o option · 62, 74  
    output · 62, 74  
    -s option · 60, 75  
    with ODBC · 56, 74  
displaying data · 60  
dj · *see reduced cost*  
dual value · 33, 75  
dummy subscript · 36  
dynamic index sets · 63  
DYNINDEX · 63

---

## ***E***

END · 35, 72  
errors  
    in models · 11, 20  
exporting  
    data · 62  
    solution · 72

---

## ***F***

fixed costs · 46  
fixed variable · 44  
for all symbol · 42  
FOR loop · 76  
FPRINT · 14, 22, 34

---

## ***G***

GENERATE · 9, 11, 35, 67, 72  
GLOBAL · 48, 81  
global entities · 79  
global statistics · 31

---

## ***I***

index sets · 63  
    dynamic · 63  
    intersection · 76  
    unions · 76  
INDICES · 63  
infeasible solution · 15  
INPUT  
    in mp-model · 20, 74  
    in mp-opt · 12, 22  
input cost · 32  
integer programming · 46, 79  
    optimising · 48  
integer variables · 47, 79

---

## ***K***

knapsack problem · 72

---

## ***L***

LET section · 36  
linear equations · 1  
linear expression · 10  
linear inequalities · 1  
list (.lst) file · 20, 61  
logical conditions · 79  
logical constraints · 48  
logical expressions · 51  
long lines in models  
    splitting · 38  
lower bound · 44

---

## **M**

MACRO section · 76  
matrix file · 20, 67, 70  
    binary · 67  
    generating · 11  
    inputting · 12, 67  
MAXIMISE · 12, 22  
model · 8  
    blending problem · 24  
    chess set problem · 11  
    coco problem - phase 1 · 30  
    coco problem - phase 2 · 38  
    coco problem - phase 3 · 44  
    coco problem with index sets · 64  
    entering a · 7, 9  
    errors in · 11, 20  
    knapsack problem · 72  
    production planning · 20  
    project planning · 83  
    restoring · 74  
model files · 17, 19  
model instance · 26, 49, 53  
model structure · 26, 49, 53  
mp-model · 3  
    starting · 9  
mp-opt · 3  
    starting · 12  
MPS file · see *matrix file*  
MPS names · 26, 67

---

## **N**

non-linear functions · 79

---

## **O**

objective function · 8, 10  
obtaining a solution · 12  
    to a MIP problem · 47  
ODBC  
    to export solution · 74  
    to import data · 55  
optimal solution · 12

optimising  
    a MIP problem · 48, 81  
    an LP problem · 12

---

## **P**

parameters · 36, 57  
partial integer variables · 79  
PRINT  
    in mp-model · 21, 60  
    in mp-opt · 13, 31  
problem files · 17  
problem statistics · 30  
problems  
    blending problem · 22  
    chess set problem · 7  
    coco · 27  
    knapsack · 72  
    production planning · 17  
    project planning · 81  
production planning · 17  
    model · 20  
project planning · 81  
    model · 83

---

## **Q**

QUIT · 11

---

## **R**

reduced cost · 32, 75  
reference row · 84  
RESTORE · 74  
restoring models · 74  
rows · see *constraints*  
rows section · 33  
running XPRESS-MP · 3

---

## **S**

secondary data · 37  
semi-continuous variables · 79

---

separating model structure · 49, 53  
SET DYNINDEX · 63  
SET UPDIR · 75  
SETS section · 84  
shadow price · see *dual value*  
slack value · 33, 75  
solution  
    chess set problem · 13  
    coco problem - phase 1 · 31  
    exporting · 72  
    saving to a file · 13, 14, 72  
    viewing · 13, 14, 31  
sparse data format · 59  
sparse data tables · 59, 62, 63  
sparse variables · 75  
sparsity · 59  
special ordered sets  
    reference row · 84  
    type 1 · 79, 84  
    type 2 · 79  
    why use? · 84  
splitting lines in models · 38  
spreadsheet connection · 55, 74  
SUM  
    in ASSIGN section · 38  
    in CONSTRAINT section · 36

---

## ***T***

TABLES section · 18

---

-e option · 59

---

## ***U***

unbounded solution · 14  
UPDIR · 75  
upper bound · 44

---

## ***V***

variables · 8  
    arrays of · 22, 25, 35  
    as data tables · 72  
    binary · 47, 79  
    bounds on · 44  
    conditional · 52  
    fixed · 44  
    integer · 47, 79  
    partial integer · 79  
    saving to a file · 72  
    semi continuous · 79  
    sparse · 75  
    special ordered sets · 79  
    subscripted · 22  
VARIABLES section · 9  
    -e option · 75  
viewing the solution · 13, 31  
Visual XPRESS · 3  
    solution print file · 31